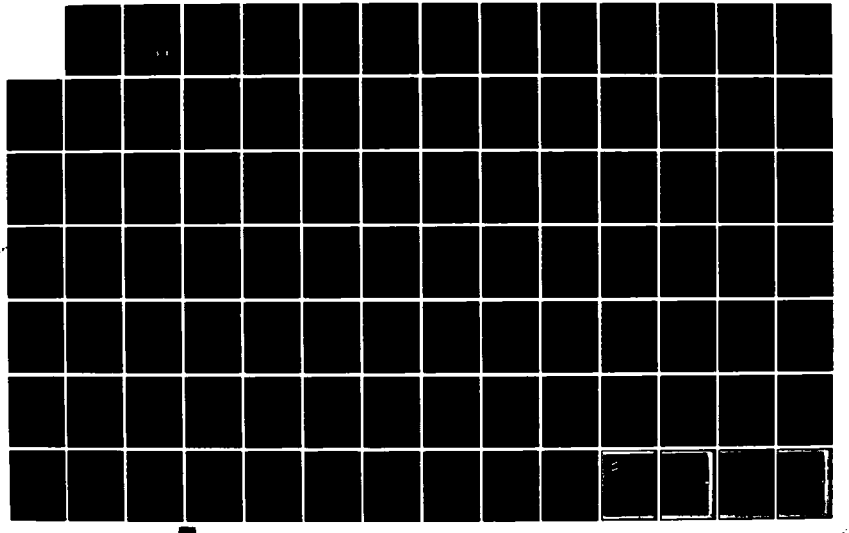
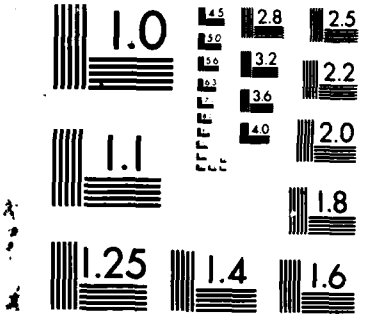
AD-A174 525 ASYNCHRONOUS DISCRETE CONTROL OF CONTINUOUS PROCESSES (U) NORTHEASTERN UNIV BOSTON MA M E KALISKI 24 FEB 86 AFOSR-TR-86-2052 F49620-82-C-0080 1/3

UNCLASSIFIED F/G 9/4 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(2)

1

AFOSR-TR- 86-2052

Asynchronous Discrete Control of Continuous Processes

Martin E. Kaliski

FINAL TECHNICAL REPORT

Contract F49620-82-C-0080

Prepared by: Northeastern University
360 Huntington Avenue
Boston, MA 02115

Prepared for: Mathematical and Information Sciences
Directorate
Air Force Office of Scientific Research
Bolling AFB
Washington, DC 20332

Approved for public release;
distribution unlimited.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD-A174 525

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: N/A
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 86-2052
6a. NAME OF PERFORMING ORGANIZATION: Northeastern University; 6b. OFFICE SYMBOL (If applicable): 423-LA	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code): 360 Huntington Avenue, Boston, MA 02115	7b. ADDRESS (City, State and ZIP Code): Bolling Air Force Base, Washington, DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-82-C-0080
8c. ADDRESS (City, State and ZIP Code): Bolling Air Force Base, Washington, DC 20332-6448	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A1	

11. TITLE (Include Security Classification): Asynchronous Discrete Control of Continuous Processes

12. PERSONAL AUTHOR(S): Martin E. Kaliski

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM 7/1/82 TO 12/31/85	2/24/86	201

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	coding, automata theory, discrete control, switching theory, feedback control

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This research concerns the analysis and synthesis of asynchronous systems that contain discrete-state feedback compensators for continuous state plants. New tools for characterizing both the interfaces and the signals present in these intrinsically hybrid systems have been developed. Generalizations of automata theory to real number alphabets have been pursued and the application of semigroup theory to the dynamics of such systems has allowed for novel approaches for characterizing the internal state of asynchronous systems to be derived. A simulator for these "asynchronous machines" has been written and allows us to bridge the gap between ideal systems, on one hand, and systems with physically constrained processing times, on the other.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Dr. Robert Buchal	(202) 767-4939	NM

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

Table of Contents

1. The Objectives of the Research Program

The broad objective of this research has been to derive mathematical analysis and design techniques for asynchronous dynamic systems with hybrid (continuous/discrete) state spaces, with specific application to the synthesis of finite-state controllers for continuous state plants. Long term applications to such application intensive problems as robotics or manufacturing engineering have formed a motivating tool for these efforts.

This research program has had a long history and, prior to this contract, was funded by AFSC under contract number F49620-80-C-0002. This prior contract focused on essentially the same issues, but for synchronous systems. As with the prior contract the current research program has been characterized by a need to re-examine certain fundamental concepts of system theory in a non-traditional setting. This has, at times, appeared to have slowed down the pursuit of its principal research objectives, as well as hamper the publication of its results. Nonetheless, a firm theoretical groundwork has been laid for future research, and this groundwork comprises a creative fusion of ideas that is both innovative and significant.

The typical finite-state control environment, often realized by a microprocessor-based controller and analog-to-digital, and digital-to-analog converters, is a hybrid environment. The state space of the plant is usually continuous in nature, and the plant's state space may indeed evolve continuously in time as well. The controller, by its very nature, operates over a finite state space, with a discrete evolution of this state in time. When the control action takes place asynchronously, driven by the occurence of external events, the times of event occurence themselves enter into the system modeling problem.

The convenience of developing software for microprocessors has led to a proliferation of ad hoc methods for designing controllers, and, worse, has led to a false sense of security about correcting easily any problems that might arise in their performance. Such problems certainly do exist, ranging from "straightforward" ones such as accuracy problems arising from roundoff errors, to more subtle ones involving pseudo-chaotic behavior and other cycling phenomena in the plant's state space. This project has viewed the absence of a fundamental characterization of the dynamics of these hybrid systems as the underlying cause for these problems. Thus a call for fundamental system-theoretic research was made.

In broadest terms this research program has thus been aimed towards modeling and understanding the nature of

asynchronous hybrid feedback systems. It has been concerned with addressing three general, related questions:

Question 1: How does one characterize the (hybrid) finite-state controller as a dynamic system, operating asynchronously in time?

Question 2: How does one model the interfaces between the plant, on the one hand, and the compensator, on the other hand? (We term these the coder and the decoder)

Question 3: How does one use this acquired understanding to effectively design compensators directly, and not indirectly as finite approximations to more conventional continuous compensators?

To address these questions new tools for modeling and analysis had to be developed, and were. These tools involve generalizations of classical automata theory to real number alphabets and the application of semigroup theory to asynchronous dynamics. The results of these endeavors are discussed in Section 2.

As the project unfolded it became apparent that most of the effort would of necessity be directed towards addressing the first two of these questions. They had to be addressed first, and lack of time prevented substantial progress from being made on Question 3.

The research into answering Questions 1 and 2 has uncovered exciting areas of general research in hybrid system theory and in the theory of asynchronous systems.

2. Status of the Research Program

2.0 Introduction

The fundamental goals of the research program have been stated above. It became apparent quite early on that much, if not all of the time, would of necessity be spent in developing key concepts as opposed to designing in detail specific controllers. The research efforts in this respect must be regarded as being successful. Key structures and models in the hybrid system environment have been identified. A firm groundwork for future work in this area has been laid because of this. Much of this work has been published, submitted for publication, or presented at professional meetings. References to this material replace substantive discussion below, since copies of this material have either been sent, or will be sent concurrently with this document (as Appendices), to the Program Manager.

2.1 Simple Asynchronous Machines

In dealing with asynchronous finite-state controllers it is necessary to understand asynchronous finite automata from a viewpoint that is both fundamental and general. Traditional approaches to characterizing asynchronous finite-state automata have generally been concerned with the state transitions of such systems (and problems related to such transitions, such as races), but not with a fundamental analysis of the evolution of the state as a finite-valued function in time. These issues were addressed in a direct manner wherein the role of event occurence time is explicit, and wherein physical contraints can be imposed in a direct, manageable way.

2.1.1 The Model

As detailed in one of our recent papers [1] the simple asynchronous machine (SAM) is a particular model of a more general finite automaton based upon semigroup properties that explicitly deals with the time of an event as well as its effect on the future behavior of the machine. It consists of a finite number of digital function generators (DFGs) that are called into play as input changes occur. These generators model asynchronous patterns of state values, and play a role in SAM characterization analogous to the impulse response functions of linear system theory. The SAM model makes minimal a priori assumptions concerning temporal spacing between events or continuity of system state at times of input transition. This generality allows

us to address fundamental realization issues for SAMs, which has been done and documented [1,11] .

The key concept in the SAM model is that each input transition triggers a "cascade" of subsequent state transitions which is pre-determined by the machine structure -- these cascades are embodied in the DFGs. The cascades may be overwritten by cascades due to subsequent input transitions, so that the output timing reflects a sequence of cascades keyed to input transition times. Various architectural configurations for generating the fundamental state transition sequences were explored and are described in the cited references. The "hybrid stack" model of the DFG is used in the simulator described in the following section.

2.1.2 The Simulator

It was important to validate the utility of the SAM model, for abstraction merely for the sake of abstraction is of little value. A simulator was developed for this purpose and is detailed in the Appendix of this report. Building upon an earlier incomplete version written in Pascal, the working simulator is written in GWBASIC for the AT&T 6300 Personal Computer and exploits the powerful graphics of this version of Basic to allow meaningful examples of SAMs to be developed, simulated and displayed. The hybrid stack model of the DFG is used in this simulator and it has been run on a variety of examples. Its use in modeling the behavior of a ripple counter and a UART are reported as representative examples in the Appendix to this report. A copy of the User's Manual, Programmer's Manual, with listings, as well as a diskette, accompany this report.

The current simulator is limited to single input, single output systems. Work to extend the simulator's capabilities to handle multiple input, multiple output SAMs is currently under way, as part of a Master's thesis project. This thesis will be completed by June, 1986, and a copy will be sent to AFSC upon its completion. (See Section 4. for further details.) The multiple input SAM model opens up the door for some important practical questions involving how the SAM is to process transitions on its several input lines that are nearly, but not exactly, concurrent.

2.1.3 Physical Constraint Modeling

The essential strength of the SAM formulation is in its ability to model the kinds of physical constraints typically present in the discrete-control environment: minimal signal processing times, minimum intervals between events, minimal duration for compensator inputs and outputs, and so on. Through the imposition of such constraints on individual digital function generators and on the class of

digital function generators as a whole, one can begin to incoporate these constraints explicitly, rather than implicitly, into the SAM model [11]. Work in this area has just begun. Current research is underway, as part of the above referenced Master's Thesis project.

It is important that such constraints be explicit rather than implicit. To understand the most general interconnections of SAMs, including specifically feedback connections, it is most important that one not limit in advance implicitly the rapid state transition rates that can arise in such configurations. Rather one must be able to model them theoretically and then re-examine them in the light of realistic physical processing constraints.

Once one has addressed the issue of physical constraint modeling attention can turn to such related significant issues of how to synthesize SAMs using commercially available components and how to employ modern techniques of design automation to expedite such syntheses.

2.1.4 Real-Time Multitasking Systems

The representation of asynchronous multitasking systems offers a specific example of the phenomena modelled by the more abstract asynchronous machines considered above. [5] Here the interest is in in process synchronization by means of an appropriate communication channel. Early work in this research program addressed such issues; they are explored in more detail in the cited reference.

2.2 Asynchronous Time Signals

In the asynchronous discrete-control setting it is easy to see that the coder inputs and outputs are, in effect, piecewise-constant time signals, with transition times between pieces occuring at event-driven times. It was felt that an understanding of this class of time functions was essential to characterizing the overall dynamics of the asynchronous control system.

Research activity in this area assumed several forms, and ties in with the aforementioned simple asynchronous machine models, for which signal ranges are assumed to be finite. In the discussion below the acronym "ATF" stands for "asynchronous time-function".

2.2.1 Generative Models

These efforts here focused on the development of generative models for the class of ATFs. Two approaches to this modelling problem were pursued. In the first approach,

when the range space of the functions is a finite set, the digital function generator (DFG) was proposed as a general purpose scheme for generating asynchronous functions of finite range. The role of the DFG in the architecture of the simple asynchronous machine has already been noted. As a "stand alone" device its implementation is of particular interest. The DFG may be implemented by using either a "hybrid stack" architecture or by using a set of independently recursively generated breakpoints [1].

A more general mechanism for generating ATFs with arbitrary ranges was developed [8,10]. This scheme consists architecturally of a value generator, a switching time generator, and a signal synthesizer. The value generator produces the range values assumed by an ATF; the switching time generator generates the breakpoints (transition times) of the function, and the signal synthesizer then outputs the resulting asynchronous waveform in real-time. This general model is a recent one that was created towards the end of the contract period in an effort to encapsulate under one umbrella both the various DFG models earlier discussed, and the more general non-finite-range class of signals, and to allow for general physically-imposed constraints to be placed upon ATFs. It also is general enough to allow for the generation of stochastic asynchronous time signals.

2.2.2 Recursive Models

Some consideration was given to the distribution of the transition points of ATFs, since, particularly in the case of recursively generated DFGs, there is an ergodic flavor to this problem. Previously unfunded work of Dr. Kaliski was extended to develop more precise notions of orbital behavior and distributions, notions that might prove useful in understanding and characterizing transition point distribution somewhat better [3,4].

The key theme of the above developments is that the transition times may be viewed as generated recursively under the iterates of an appropriate timing generator. This ties in as well with the above cited themes of independently recursively generated breakpoints.

2.2.3 Physical Constraint Modeling

The essential problem, from the point-of-view of modelling realistic plants and controllers, is to identify and model various physical constraints that fall upon the plant-coder-controller-decoder ensemble. Physical constraints manifest themselves in a variety of ways, particularly when the issue of asynchronous systems and transition times are discussed. Physical systems cannot instantaneously respond to signal changes, nor, it may be argued, can they recognize signal changes that occur too rapidly. They seem to require a minimal "energy content" in signals to respond to them, and thus not only are signal

amplitudes, but signal breakpoint distributions, constrained.

Efforts have begun to grapple with this problem and the continuing research efforts of Northeastern graduate students will address these issues. When resolved one may in turn apply such models to the above described generative schemes for ATFs.

2.3 Asynchronous Coders

In the asynchronous control setting plant outputs are sampled at irregular, event-driven times. A fundamental question of this research program has been to understand how to incorporate such times in the coder (and decoder) models successfully developed for the synchronous case under the earlier project funding . The general problem of modeling these interface signals has been described above. In what follows discussion turns to how the interfaces themselves can incorporate time. The view that asynchronous coder design is but an extension of synchronous coder design into a higher-dimensional space is postulated.

Note that much of this work thus extends the previously developed theory of coder design, particularly the decomposition results for such coders [2] and the related acceptor models. Also the viewpoint that coders in effect transform sequences of points with real-valued coordinates into sequences from a finite alphabet still is a valid one.

The research program did not deal with the issue of decoders to any great extent. One the one hand, their structure is far simpler than that of the coder, since they map finite spaces into non-denumerable ones; on the other hand, there are some very interesting practical issues in developing decoders that are optimal, according to various criteria.

2.3.1 Incorporating Time into the Coder Model

By treating event occurence time as an additional coder input -- one which always increases, of course -- one is able to model asynchronous coders as special types of partially-specified automata defined over real number alphabets [2,4,7,9,12,13]. This viewpoint is a profitable one, for it allows the derivation of necessary and sufficient conditions for the finite-state realizability of asynchronous coders. One terms coders so-realizable as finitary.

The derived theory is an extension of the previously developed theory for synchronous coders, as noted, and revolves around the concept of tightly structured input/output maps. This opens up an exciting new area of

research, which one can term invariant theory for finite automata (section 2.3.4) .

2.3.2 Finitary Asynchronous Coders

Finitary coders are important architecturally since their use augments the complexity of the finite state part of the control system and not the continuous-state part [2]. Such coders are always realizable as a cascade of a memoryless quantizer (an "ordinary" analog-to-digital converter) and a finite state machine.

It was the recognition that event-time serves to simply increase the dimensionality of a coder that forms the central theoretical accomplishment of this phase of the research. By using sample-and-hold elements to "capture" event times the asynchronous coder can be synthesized using the tools of synchronous coder theory.

2.3.3 Linguistic Models for Coders

Coders, as noted, may be viewed as devices that transform strings in one alphabet (a non-denumerable one) into strings in another alphabet. At a conceptual level, one is then able to apply generalized tools of formal language theory to their characterization [2]. This view of the problem is an interesting one in the search for canonical forms for coders. Time did not permit, however, its full exploration. By analogy with the language hierarchies of formal language theory, one may define coders of increasing complexity and power. This, too, remains an incomplete area of research.

2.3.4 Invariants in Finite-Automata Theory

It was observed that many of the ideas of finite automata theory do not of necessity depend upon both the state space and the input alphabet being finite. For example, the concept of Nerode equivalence [2] can be utilized to define finite state realizations for coders, although the input alphabet is non-denumerable. Similarly concepts of non-determinism in automata, and formal language equivalents do not depend upon the finiteness of the input alphabet.

This opens the door to some basic research in abstract automata theory. Since many techniques of discrete-controller design indeed revolve around the concept of one automaton controlling another, the need for such abstraction is justified. These topics are still being investigated as of this writing.

2.4 Topics Not Completely Addressed

Several proposed topic areas were not addressed due to a lack of time. This was a result of conscious decision on the part of the principal investigator(s).

2.4.1 Finite-State Controller Design

As already noted, time did not permit a direct solution to this problem in the asynchronous case. As discussed in section 5, an independent activity for designing controllers in the synchronous case, for discrete-time plants, is currently under way. By choosing the route pursued in this research program -- that of laying fundamental groundwork -- it is apparent that one principal goal for the future must be to tie together these separate efforts to effect the design of meaningful controllers.

2.4.2 Characterizations of Coder Behavior

One interesting area of research proposed was to try to develop both qualitative and quantitative schemes for modeling the behavior of coders and to deal with such issues as coder similarity, and practical coder realizations. Certainly the finitary coder model is a beginning step in this overall goal. More definitive measures must be defined, however, to allow one to talk about coders that satisify various design and implementation criteria.

2.4.3 Feedback Connections of SAMs; Stability

Having explored the SAM model as a stand-alone device it is natural to deal with such questions as interconnections of SAMs. Such structures may be series connections, parallel connections, or, more importantly, feedback connections. When SAMs are connected in feedback transitions may occur at times that are extremely close together. It has been postulated that the SAM model is general enough to be closed under feedback connections, nonetheless. Future research is needed to resolve this issue, as well the general issue of stability of SAMs, from both a definitional and applications point-of-view.

2.4.4 SAMs and Quantized Continuous Systems

One interesting question that naturally evolves out of this research effort is the following one: look at the decoder-plant-coder ensemble, in that order, as a dynamic system. Its inputs and outputs are sequences in a finite set. Can we view this system as equivalent to a SAM? This, in part, is tantamount to answering the related question of under what conditions this mapping from input sequences to

output sequences is finite-state realizable. These are interesting questions that are very significant ones, for knowing that the decoder-plant-coder ensemble is effectively a SAM reduces the question of compensator design to one of controlling one SAM with another.

3. Publications

The following works have been published and/or submitted for publication during the term of this contract. The order is in each category is alphabetical, by name of first author. The list is categorized into three categories: papers accepted by or submitted to technical journals, papers presented at conferences (and appearing in conference proceedings), and papers written as unpublished internal research memoranda.

Papers Submitted to/Accepted by Technical Journals

1) Johnson, T.L. and Kaliski, M.E., " Realization of Finite-State Asynchronous Machines, " submitted to IEEE Transactions on Automatic Control, April, 1983. Currently under revision. (Abridged version presented at 22nd IEEE Conference on Decision and Control, December, 1983, San Antonio, TX)

2) Kaliski, M.E., " Finitary Coders: The Interfaces in Finite-State Compensation Schemes, " submitted to IEEE Transactions on Automatic Control, May, 1985. Currently under revision.

3) Kaliski, M.E. and Klein, Q.L., "Behavior of a Class of Nonlinear Discrete-Time Systems, " Journal of Computer and System Sciences, Vol.31, No. 1, August, 1985

4) Kaliski, M.E., Kwankam, S.Y., Halpern, P. and Shulman, D., "A Theory of Orbital Behavior in a Class of Nonlinear Systems: Chaos and a Signature-Based Approach, " accepted for publication by Journal of Computer and System Sciences, October, 1985. To appear.

Papers Presented at Conferences

5) Johnson, T.L., "Multitask Control of Distributed Processes," presented at 22nd IEEE Conference on Decision and Control, December, 1983, San Antonio, TX.

6) Kaliski, M.E., " On Realizations of Partially-Specified Input/Output Maps by Finite Automata, " presented at 1984 ACM Computer Science Conference, February, 1984, Philadelphia, PA.

7) Kaliski, M.E., " Finite Automata Over Real Number Alphabets: Some Theoretical Results and Applications, " presented at 1985 ACM Computer Science Conference, March, 1985, New Orleans, LA.

8) Kaliski, M.E. and Kwankam, S.Y., " Asynchronous Real-Time Coders in the Discrete Control Environment: Generative Models for Input/Output Spaces, " presented at 23rd IEEE Conference on Decision and Control, December, 1984, Las Vegas, NV.

9) Kaliski, M.E. and Wimpey, D.G., "Towards a Theory of Asynchronous , Real-Time Coders and Their Applications to Discrete-Control of Continuous Processes, " presented at 1983 American Control Conference, June, 1983, San Francisco, CA.

10) Kwankam, S.Y. and Kaliski, M.E., " A Generative Model for Asynchronous Finite-Valued Time Signals as Applied to Computer-Based Process Control, " accepted for presentation at the IFAC Symposium on Microcomputer Application in Process Control, July, 1986, Istanbul, TURKEY.

11) Kwankam, S.Y., Kaliski, M.E., and Johnson, T.L., "Asynchronous Finite State Machines: Simulations with Imposed Processing Constraints, " accepted at 1986 American Control Conference, June, 1986, Seattle, WA.

Internal Research Memoranda

12) Kaliski, M.E., "Extensions of Partially-Specified Automata Incorporating Time as an Input, " Northeastern University Internal Memorandum, September, 1983.

13) Kaliski, M.E., " Towards a Theory of Finitary Asynchronous Coders, " Northeastern University Internal Memorandum, January, 1983.

4. Personnel

The following professional personnel have been the primary personnel associated with this project since its beginning in July, 1982:

Dr. Martin E. Kaliski, Professor of Electrical and Computer Engineering, Northeastern University, Boston, MA has been a co-principal investigator and, since July 1984, principal investigator on this research contract.

Dr. Timothy L. Johnson, Control Technology Branch, General Electric Corporate Research and Development, Schenectady, NY was, while he was still with Bolt, Beranek, and Newman, Inc. Cambridge, MA, a co-principal investigator on this project. He has remained involved in an unfunded capacity since he assumed his duties at General Electric in July, 1984.

Dr. David G. Wimpey, a former doctoral student of Dr. Johnson's (and supported under our previous contract) was associated in an unfunded capacity with this project while a faculty member at Northeastern University in the Department of Electrical and Computer Engineering, from July, 1982 until his return to the Republic of South Africa in September, 1983. His principal contributions were in the area of coder and compensator design.

Dr. S.Y. Kwankam, a former doctoral student of Professor Kaliski's, at Northeastern, has remained involved in an informal capacity with this program since his return to the University of Yaounde, in Cameroon in 1979. He spent the summer of 1985 working on this project as a funded Fulbright Scholar. His contributions were in the areas of asynchronous machines and asynchronous time signals.

The group of professional personnel listed below have played a smaller, but nonetheless important role, in the specific project areas listed below. Their work has been done in an unfunded capacity.

Dr. Karen A. Lemone, a former doctoral student of Professor Kaliski's, at Northeastern, is currently an Associate Professor in the Department of Computer Science at Worcester Polytechnic Institute, Worcester, MA. She has contributed to the development of conceptual issues in languages defined over real number alhpabets, with applications to coder design.

Quentin L. Klein, of W. Newton, MA, has worked with Professor Kaliski in the study of orbital behavior in one-dimensional nonlinear systems.

Pamela Halpern, of Comp-All Systems, Lynnfield, MA, is a former doctoral student of Professor Kaliski's at Northeastern. She, too, has worked in the area of orbital behavior studies.

David Shulman, a graduate student of Professor Kaliski's at Northeastern University, has played a role in orbital behavior studies as well.

Andrew Miller, is an American Electronics Association Fellow at Northeastern, and is currently working on his Master's thesis for Professor Kaliski. He has been active in developing the simple asynchronous machine models and simulations described earlier.

5. Interactions

While Dr. Johnson was with Bolt, Beranek, and Newman, Inc. (until June, 1984) he and Dr. Kaliski met regularly (every other week) to ensure adequate progress of the research program. With him based at General Electric Company in Schenectady, NY, since then, the level of interaction has naturally decreased somewhat. Nonetheless, regular monthly meetings which take place for an independent joint activity of Drs. Johnson and Kaliski (see below), provide a continuing forum for interaction on this project.

Research results have been disseminated at a variety of conferences around the country (section 3.), and have been submitted to publications appealing to both the computer science and control systems community. This reflects the spirit of the research as a venture into hybrid systems theory.

Dr. Kaliski serves as a consultant to General Electric Company, Corporate Research and Development, for Dr. Johnson, in the general area of finite-state synchronous controller algorithm design and implementation. This problem represents the practical, computational end of the general research area of this project. There, of course, has been great care taken to avoid any potential compromise between the general research goals of this project and those of that specific project. The essential point is that this parallel activity has served to both keep interactions in this entire area at a healthy level and has provided means for validating theoretical constructs in a practical context.

6. Appendices

The following Appendices contain material not previously submitted to AFSC. In addition to this material a single diskette containing programs and data files for the SAM simulator is being sent (along with instructions for its use) with this report.

6.1 The Sam Simulator

The following supplementary material concerning the SAM simulator is contained in this section:

a) User's Manual

b) Modeling of 74LS93 Binary Counter

c) Modeling of RCA CDP1854 Programmable UART

d) Programmer's Manual

e) Program Listings

6.2 Additional Publications

Copies of publications 3 and 4 are being submitted with this report, as they were not previously sent to AFSC in final form.

6.1 The SAM Simulator

a) User's Manual

TABLE OF CONTENTS

1.0 INTRODUCTION

This User's Manual explains the operation of the asynchronous machine simulator which runs under GW-Basic. Since the simulator is menu-driven, most of the information is self-explanatory. This manual further clarifies these menus and assists the user in correctly working with the simulator.

It is assumed that the user of this manual is familiar with the paper, "Realization of Asynchronous Finite-State Machines" by T.L. Johnson and M.E. Kaliski. This simulator is essentially a direct implementation of some of the ideas set forth in that paper.

To further clarify the information included in this manual, refer to two examples included which are executed using this simulator, referred to as SIMSAM (SIMulator for Simple Asynchronous Machines). One example is the simulation of a simple binary counter. The other is a more complex example of a Programmable UART.

The following files must be included on the drive designated in the simulation program as the default drive:

MAIN.BAS
INPOVR.BAS
DFGOVR.BAS
LGFOVR.BAS
RSMOVR.BAS
PLTOVR.BAS

See elsewhere in this manual to set the default drive.

The order of information presented in this User's Manual is consistent with the order in which information appears in the execution of the SIMSAM program.

A note regarding program output:
All text displayed by the simulator program will be indented in this User's Manual

2.0 GETTING STARTED

The SIMSAM simulator can be run on any IBM compatible pc with 256k bytes of memory and GW-Basic (or equivalent BASIC interpreter) running under MS-DOS.

After loading GW-Basic, load in the main module by typing:

LOAD "dd:MAIN"

where "dd" is a valid drive specification for the location of the file "MAIN.BAS".

Now type "RUN" to execute the main module under GW-Basic.

After a short period of time during which the title of the program is displayed, the following menu is displayed:

S I M S A M M E N U

1 -- Enter Input Waveform
2 -- Enter Digital Function Generators
3 -- Enter Logic Function Generator
4 -- Run simulation
5 -- Plot results
6 -- Set default drive
7 -- Exit

Enter selection by number ?

This menu permits one to make a selection by simply typing the number of the selection and hitting the Return key. A brief description of each of the selections is given here. If an invalid selection is made, the simulator emits a short beep and the menu is redisplayed.

1 -- Enter Input Waveform

This choice displays another menu to enter the values of an input waveform, either from a file or the terminal. This waveform can be edited while in memory, and can be saved on the disk.

2 -- Enter Digital Function Generators

This choice displays a menu to enter the values of digital function generators. As with the input waveform, one can load the DFGs from the terminal or a file, and perform simple editing.

3 -- Enter Logic Function Generator

This choice displays a menu to enter the values of a logic function generator.

4 -- Run simulation

This choice allows one to execute a simulation based on the data entered from choices 1, 2, and 3 of the SIMSAM MENU. The simulator will not function correctly if choice 4 is made before choices 1-3. Choices 1-3 can be selected in any order.

5 -- Plot results

This choice gives one a graphical display of any of the waveforms, including those entered by the user and those generated by the simulator.

6 -- Set default drive

By default, the simulator sets the default drive to "B". This is the drive where all parts of the simulator program are loaded from, and also where all waveforms are saved. This choice allows one to change the default drive.

7 -- Exit

This choice causes the program to terminate, and control is returned to GW-Basic.

3.0 ENTERING INPUT WAVEFORM

If choice 1 is selected from the SIMSAM MENU, the following menu is displayed:

```
        I N P U T   W A V E F O R M   M E N U

    1 -- Load input waveform from file
    2 -- Enter input waveform from keyboard
    3 -- Modify input waveform in memory
    4 -- Exit to main menu

    Enter a selection by number ?
```

As with the SIMSAM MENU (main menu), one can make a selection by typing the corresponding number and hitting the Return key. Each of the choices is described in detail here.

3.1 -- Load input waveform from file

This choice causes the screen to be cleared and the following prompt to be displayed:

```
    Enter M (<= 8), the number of input levels
    M =
```

At this prompt enter the number of levels in the input waveform. The program checks to see that the entered value is within range.

The program then prompts for the name of the file where the input waveform is saved with the prompt:

```
    File name for input ?
```

A valid filename for the input waveform consists of "INPUTxx", where xx can be any two alphanumeric characters. The program only distinguishes between different last two alphanumeric characters for different input file names. The program loads the specified file, and verifies that the data in the file is consistent with the number of input levels entered previously.

After the file has been loaded, the INPUT WAVEFORM MENU is redisplayed. If the number of levels of the waveform read from the file is greater than the limit entered previously, the waveform must be entered from the keyboard.

3.2 -- Enter input waveform from keyboard

As with choice 1, this choice first prompts for the number of levels in the input waveform. After this, it asks for the pieces of the input waveform with the dialogue:

```
Enter input signal

Enter number of pieces, (<= 30)
?
```

At this point one should enter the number of distinct pieces comprising the input waveform. The program verifies that the upper limit is not exceeded. The program then prompts:

```
Enter waveform in the format: value and start time

piece # 1
```

Here the user should enter the level of the piece of the input waveform, and the time it starts, separated by a comma. Normally the start time begins at time 0. The program checks to see that the level value is an integer in the range previously specified, and that the start time for a successive piece is greater than for the previous piece. In other words, successive start times increase monotonically.

After all the pieces of the input waveform have been entered, the program echoes all the pieces on the screen and asks for any changes with the prompt:

```
Any changes (Y)es or (N)o
```

If a "Y" is entered here, the program prompts:

```
Enter changes in the format: piece #, value and start
time
To end enter piece # of 0
```

Simply enter the parameters asked for in the order specified, separated by commas. As with all other parts of the program, it verifies that entered data is within range.

After any and all changes have been made and the input waveform is exactly what is desired, the program prompts to see if the entered data is to be saved in a file:

```
Note that if this input function is not saved, and
the results of a run using this input are saved, such
as result file will NOT contain information on the
input function which was used.

Save input? Yes(Y) or No (CR) ?
```

If the input is not saved, the INPUT WAVEFORM MENU is redisplayed. If the input is to be saved, the program prompts:

> Enter file name under which input is to be saved in the form 'INPUTxx', where xx are alphanumeric characters. ?

After entering a valid filename, the input waveform data is saved in that file on the default drive, and the INPUT WAVEFORM MENU is redisplayed. This filename can be one which has already been used, in which case the new data will overwrite the old data in the file.

3.3 -- Modify input waveform in memory

This selection should only be chosen after loading an input waveform from memory with selection 1. The program proceeds through the same steps as detailed for choice 2 after the input waveform had been entered.

It prompts for which pieces of the waveform are to be modified, and then reminds one to save the modified waveform back out on disk.

After all modifications have been completed, the INPUT WAVEFORM MENU is redisplayed.

3.4 -- Exit to main menu

After a correct input waveform resides in memory from one of the other choices, this selection causes the SIMSAM MENU to be redisplayed to load other waveforms.

4.0 ENTERING DIGITAL FUNCTION GENERATORS

If choice 2 from the main menu is selected, the following menu is displayed:

```
                    D F G   M E N U

    1 -- Load DFG from file
    2 -- Enter DFG from keyboard
    3 -- Modify DFG in memory
    4 -- Exit to main menu

    Enter selection by number ?
```

Any of the selections on this menu can be chosen by typing the number corresponding to the selection and hitting the Return key. Each of the selections is described separately here:

4.1 -- Load DFG from file

After selecting this choice, the program prompts:

```
    Enter N (<= 9), the number of states
    N =
```

The user is expected to enter the number of distinct states for the simple asynchronous machine being simulated. The program verifies that it is within the bounds specified. It then prompts for the file name:

```
    File name from which DFGs are to be read. ?
```

Here, the file name is entered in the form "DFGxx" where xx can be any two alphanumeric characters. If the filename is found on the default drive and the number of states of the DFGs in the file are compatible with N entered previously, the DFG MENU is redisplayed.

If any problems occur, the program asks that the DFGs be entered from the keyboard. This is explained under choice 2.

4.2 -- Enter DFG from keyboard

With this choice, the program again prompts for the number of states in the DFG, and checks to see that it is less than a predetermined maximum, then prompts:

```
    Enter DFG for input level mm and state nn
    Enter number of pieces, (<= 8) ?
```

where mm is one of the input levels in the previously specified range, and nn is one of the SAM states. Note that

because the number of input levels is needed, an input waveform must have been entered prior to entering DFGs.

For each DFG, the program needs to know how many pieces comprise the DFG for a particular input level and machine state. After that, the program prompts:

> Enter waveform in the format: value and start time
> piece # 1

Each of the pieces of a particular DFG are entered with the value and start time separated by a comma. The program checks that the value is an integer and one of the possible state values, and that the start time is greater than zero, and is monotonically increasing for subsequent pieces.

This continues for all possible combinations of state values and input levels. After all DFGs have been entered, the program prompts for any edits as explained in the next selection. If there are no edits, or after the edits are complete, the DFG MENU is redisplayed.

4.3 -- Modify DFG in memory

This choice should only be made after choice 1, or alternately, if a DFG has been entered from the keyboard, this choice is automatically made.

The program echoes each DFG and prompts for changes:

> Any changes (Y)es or (N)o ?

If there are changes they are entered in the format: piece #, value, and start time. A piece # of 0 terminates the edits for a particular DFG.

After all changes have been made for one DFG, or if no changes are necessary, the next DFG is displayed, and a request for edits is made.

After all changes have been made, the program prompts:

> Are DFGs to be saved (Y)es or No ?

If no, the DFG MENU is redisplayed. If yes, the program prompts:

> Enter file name under which DFG is to be saved ?

The file name should be of the form "DFGxx" where xx is any two alphanumeric characters. If the filename already exists on the default drive, the contents of the file are overwritten. The DFG MENU is then redisplayed.

4.4 -- Exit to main menu

This selection causes the SIMSAM MENU to be displayed.

5.0 ENTERING LOGIC FUNCTION GENERATOR

If choice 3 from the main menu is selected, the following menu is displayed:

L F M E N U

1 -- Load LF from file
2 -- Enter LF from keyboard
3 -- Modify LF in memory
4 -- Exit to main menu

Enter selection by number ?

Any of the displayed menu items can be chosen by selecting the appropriate number and hitting the Return key. Each of the selections is detailed here.

5.1 -- Load LF from file

This selection first prompts for the filename of the logic function generator:

File name from which LF is to be read ?

The file name should be of the form "LFxx" where xx is any two alphanumeric characters. This file is read from the default drive. The program checks that the dimensions of the LF are consistent with the number of states of the machine and the number of input levels. Because it uses this information, the input waveform and the DFGs must be entered before the LFs.

After the file is read, the LF MENU is redisplayed.

5.2 -- Enter LF from keyboard

After choosing this selection, the program displays the prompt:

Enter number of output levels ?

This answer must be the number of distinct levels in the output signal.

The dialogue continues:

Enter ns values for output corresponding to input level nn and each of the ns states.

The user is directed, for a particular input level, to specify what the output level is for each of the states of the machine. (ns represents the number of states). This is

repeated for each of the nn input levels. The program checks that the output level does not exceed the number of levels specified earlier.

After all output levels have been entered, the program automatically goes into edit mode which is identical to LF MENU selection number 3 and will be explained there.

5.3 -- Modify LF in memory

This selection should only be made after loading a LF from a file, or control will automatically be passed to this point after entering a LF from the keyboard. The program prompts:

Readout map just entered is as follows:

Input	State	Output
.	.	.
.	.	.
.	.	.

The output level is displayed for each of the machine states and for one particular input level. The user is then asked if any changes are necessary. If they are the program prompts:

Enter changes in the format, state and output.
To end, enter state value of 0.

The program verifies all corrections to the data. After all output levels have been modified, if necessary, the program prompts for a file name to save the changes:

Is LF to be saved (Y)es or No ?

Enter file name under which LF is to be saved. ?

This file name must be of the form "LFxx" where xx can be any two alphanumeric characters. If the filename already exists on the default drive, the contents of the file are overwritten. After the LF is saved, the LF MENU is redisplayed. The user should be aware that if changes to an LF are not saved, they cannot be recaptured later.

5.4 -- Exit to main menu

This selection causes the SIMSAM MENU to be redisplayed.

6.0 RUN SIMULATION

If choice 4 is selected from the main menu, this causes the simulator to run using the input waveform, DFGs and LFs entered previously. Unlike the other main menu selections, this one does not cause another menu to be displayed. The program prompts for necessary information:

```
Maximum # of system events in simulation (<= 240)
no. of events = ?
```

The simulator is asking for the maximum number of events to record in the simulation. If the actual number of events is less than this amount, there is no problem. If the actual number is greater, the simulation terminates when the maximum number has been reached. This prevents the possibility of an infinite number of events in the case where the simulated machine oscillates.

The simulator next prompts for a physical processing time. By hitting the Return key, a default value of 0 is assumed, but any other value can be chosen. This value can be used to simulate the latency of a particular machine.

The program then prompts:

```
Ready to run simulation. Enter initial state,
which must be an integer between 1 and ns
```

where ns is the number of states of the machine. The program ensures that the initial state is within range, then runs the simulation. At the completion of the simulation, all system events are listed in the form:

Index	Output value	Start time	Input	State
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

The index is an integer starting at 1 and is incremented for each change in the input or state. The output value is simply the combination of the input level and the state as specified in the logic function generator. The start time is the time when the change detected by the simulator begins. Again, the change can recognized either in the input or the state.

After all system events have been listed, the program prompts to see if the results of the simulation are to be saved in a file:

```
File name for saving results ?
```

A file name of the form "RESxx" should be entered, where xx can be any two alphanumeric characters. If the results are not to be saved, simply hit the Return key.

After saving the results in a file, the SIMSAM MENU is redisplayed.

7.0 PLOTTING WAVEFORMS

If selection 5 is chosen from the main menu, another menu is displayed:

```
                    P L O T   M E N U

     1 -- Plot input function
     2 -- Plot state function
     3 -- Plot output function
     4 -- Plot DFGs
     5 -- Load results from file to plot
     6 -- Exit to main menu

     Enter selection by number ?
```

Each of the menu items is described here in detail. For examples of any of the plots, refer to the Programmable UART example or the Binary Counter example elsewhere in this Appendix. Be sure to load the appropriate files as explained below before attempting to plot any functions, or errors might result and possibly terminate the program.

7.1 -- Plot input function

Choosing this selection causes the input waveform in memory to be displayed in graphic form on the monitor in high-resolution mode. The input levels are displayed on the y-axis and units of time are displayed on the x-axis. The x-axis will be the same for all plots drawn and is derived from the event times of the simulation. Because of this, the input function and DFGs cannot be plotted before a simulation run (or before the results of a previous run have been loaded with menu selection 5). This plot will be labeled "INPUT FUNCTION". Hit the Esc key to return to the PLOT MENU.

7.2 -- Plot state function

Choosing this selection causes the state evolution as a function of time to be plotted. This state function is obtained from the simulation run, or by loading the results file with menu selection 5.. The x-axis is the event time and the y-axis records the state levels. This plot will be labeled "STATE FUNCTION" and will also list the names of the input, DFG, and LF files as well as the initial state of the machine and the processing time. If certain data had not been saved or retrieved from a file, the filename for that data would be left blank. After examining the plot, hit the Esc key to return to the PLOT MENU.

7.3 -- Plot output function

This selection causes the output of the simulated machine to be displayed with the output levels on the y-axis and machine event times on the x-axis. This plot is labeled "OUTPUT FUNCTION" and provides the names of the simulation files, and the initial state and processing time. This plot can be obtained after running a simulation, or after loading a results file using menu selection 5. After examining the plot, hit the Esc key to return to the PLOT MENU.

7.4 -- Plot DFGs

This selection causes each of the mm x nn DFGs to be plotted in succession, where mm is the number of input levels, and nn is the number of states of the simulated machine. The event times are plotted on the x-axis and state levels are plotted on the y-axis. Each DFG includes a title consisting of the filename where DFGs are stored, and "Element a, b" where a is one of the possible input levels and b is the initial state for this DFG.

To see the next DFG in succession, press the Esc key. After all DFGs have been displayed, the PLOT MENU is redisplayed. The DFGs will only be displayed correctly if a simulation had been run prior to the plotting, or the input, DFG, and results files have been loaded.

7.5 -- Load results from file to plot

Selecting this choice causes the program to prompt for the results filename. This filename should be of the form "RESxx", where xx is any two alphanumeric characters. After the results have been loaded, the PLOT MENU is redisplayed.

This only loads enough information from the results file to plot the state (menu selection 2) and output function (menu selection 3). The advantage of using this menu selection is that the simulation does not have to be rerun. If this is done after a simulation run, then the results are still residing in memory and do not have to be loaded from a file with this menu option.

7.6 -- Exit to main menu

This exits the SIMSAM MENU

8.0 PRINTING PLOTS

Before being able to print plots, one must first run the program "GRAPHICS" at the MS-DOS prompt before loading GW-Basic. If this is not done, no plots can be made.

When a plot is displayed on the screen it can be printed by pressing the Shift-PrtSc key. The displayed plot will be reproduced exactly on the printer. This set of keystrokes can be repeated as often as desired.

9.0 SETTING THE DEFAULT DRIVE

Choosing selection number 6 from the main menu causes the system to prompt for the default drive to use for all subsequent file access, of both program and data files. The drive value entered must not include a colon. This value is initially set to drive B, but can be changed before loading or saving any other files. This default drive must not be changed in the middle of a simulation session.

The program does not initially check to see if the drive name is valid. It will only verify this when it first tries to access the drive to read or write a file. It is the user's responsibility to see that the drive designation exists. After obtaining the default drive information from the user the SIMSAM MENU is redisplayed.

b) Modeling of 74LS93 Binary Counter

b) Modeling of 74LS93 Binary Counter

MODELING OF 74LS93 BINARY COUNTER

To demonstrate the capabilities of the Simple Asynchronous Machine (SAM) simulator a 74LS93 4 bit binary counter is modeled. Because of current limitations of the simulator, only three of the four bits of the counter are used.

Figure 1 show a diagram of the counter chip and external connections for the configuration simulated. A binary input signal is applied to input B of the 74LS93 and the state of the counter is taken from the outputs QB, QC, and QD.

In the simulator, the input signal is described as a two level signal with a fixed period (figure 2). The counter is triggered on the falling edge of the input signal.

Since three bits of the counter are used, eight states are possible. Sixteen DFGs are needed to fully describe the behavior of this counter. These DFG waveforms are illustrated in figures 3-18. Each DFG waveform has a title "DFGDM Element x, y", where DFGDM is the filename where the waveforms are stored, x is the current input level, and y is the state. Note that the input can assume one of two binary levels, 1 or 2. The state is described by a digit from 1 to 8.

The counter is in state 1 when the QD, QC, QB outputs are 0,0,1 respectively. State 2 represents when the binary outputs are 0,1,0 and so on. State 8 is when all outputs are zero.

Since the counter is incremented only on the falling edge of the input signal, or when the input changes from level 2 to level 1, all DFGs for input level 2 are flat. That is, if the counter is in state y, and the input level is 2 the counter will remain in that state until the input level changes.

When the input level becomes 1, the binary counter with ripple carry is incremented. Because of the ripple carry and finite delay of each JK flip-flop in the counter, the least significant bit of the counter (QB) changes state before the more significant bits.

As a result, if the counter is in state 7, for example, and the input changes from level 2 to level 1, the counter will pass from state 7 to state 6 to state 4, and finally to state 8 after the carry has rippled through all flip-flops. This is shown in figure 9.

Figure 19 shows how the state of the modeled counter changes when the input signal alternates between levels 1 and 2 at regular intervals. The state is seen to progress in a

staircase fashion overall, but for certain state transitions, the counter passes briefly through some intermediate states. Again, note the transition from state 7 to state 8 as an example.

The output function is illustrated in figure 20. It is identical with the state function because the simple ripple counter modeled has no combinational circuitry at the outputs of the flip-flops. This can easily be added to the model if desired.

Although the binary counter modeled by the SAM simulator is a simple example, it demonstrates the capabilities of the simulator. With some enhancements the simulator will be able to handle more complex asynchronous circuits where the effects of inputs on the overall response are not obvious. Then the simulator will become a more useful tool.

74LS93

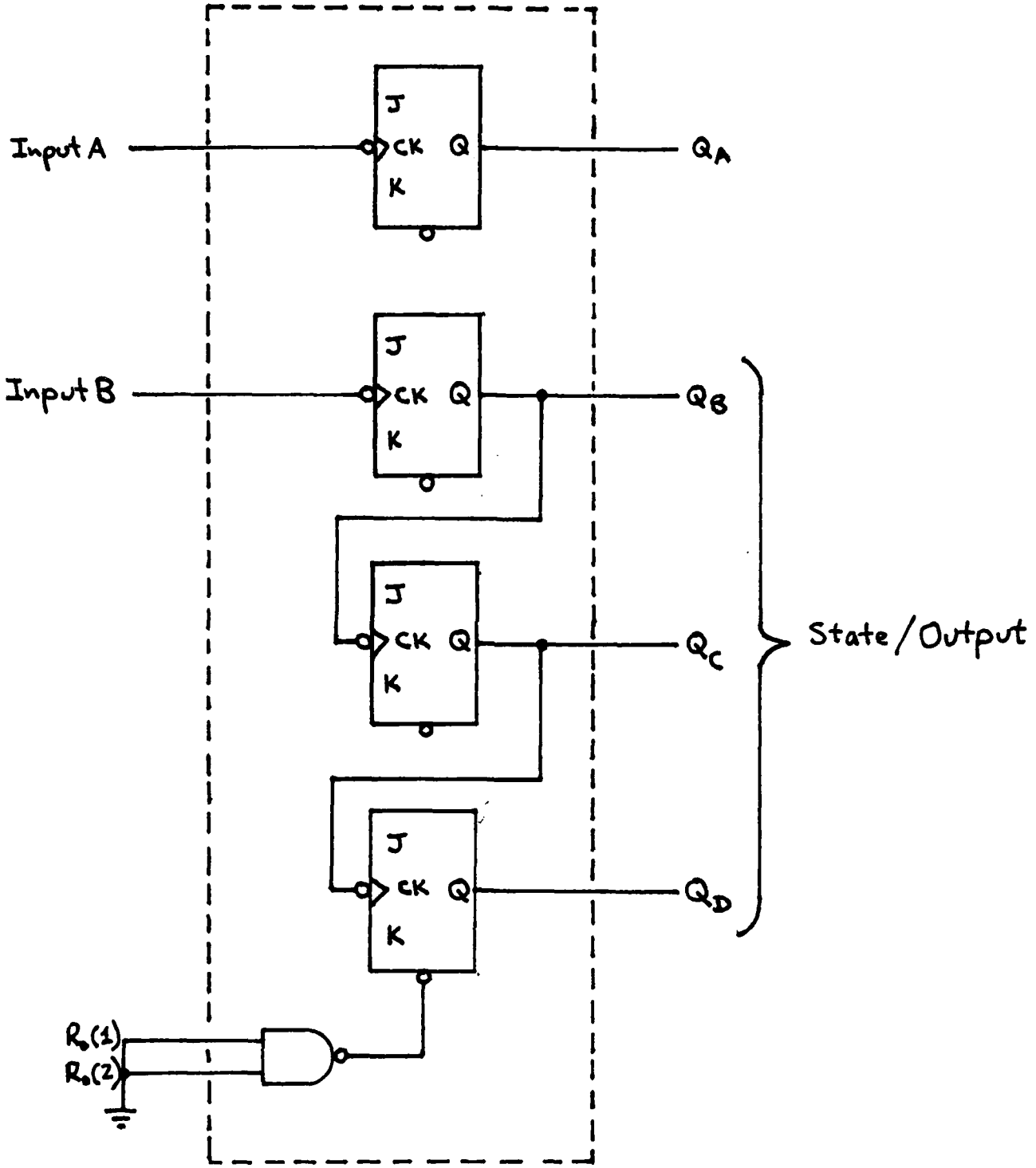


figure 1.

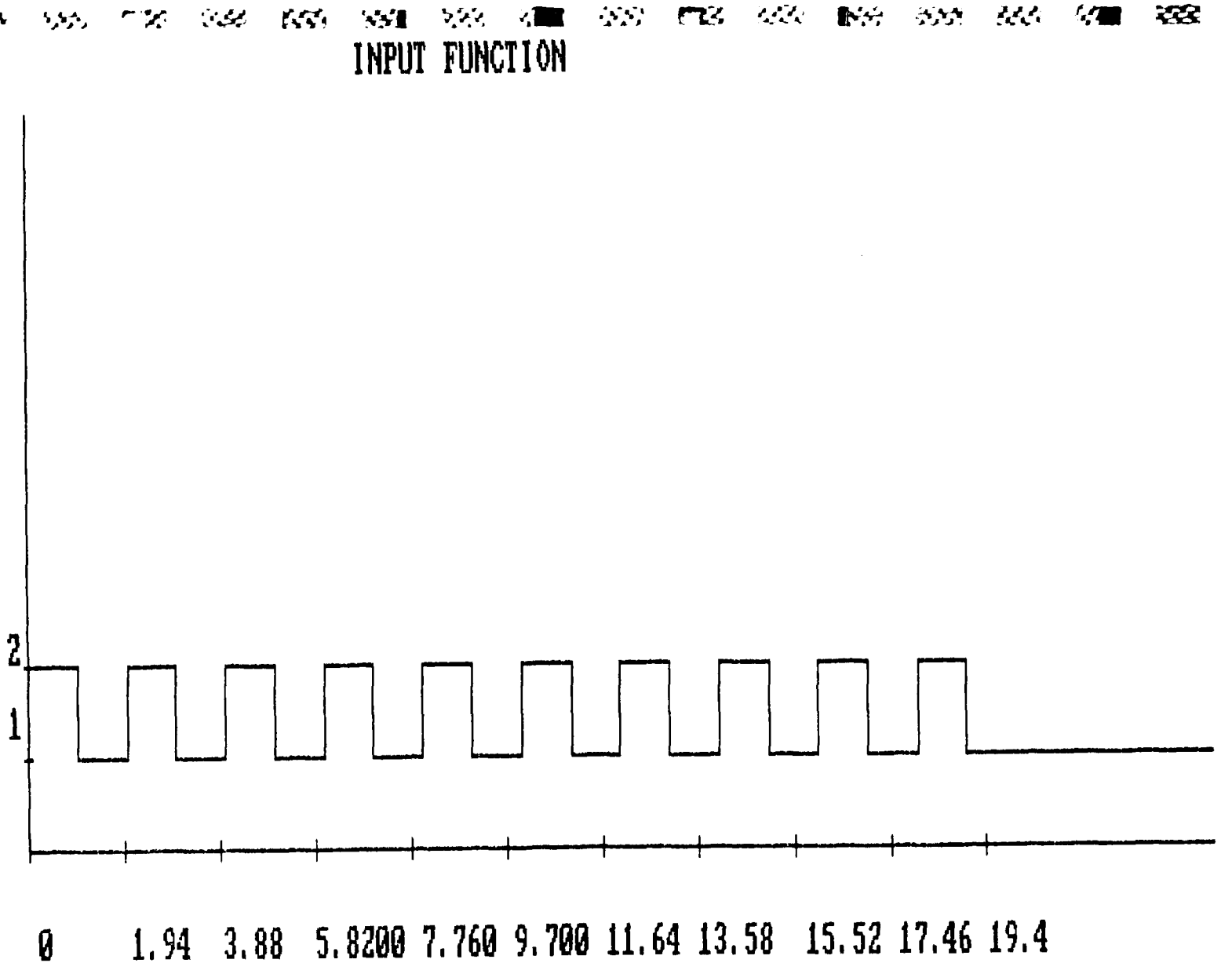

INPUT FUNCTION
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 2.

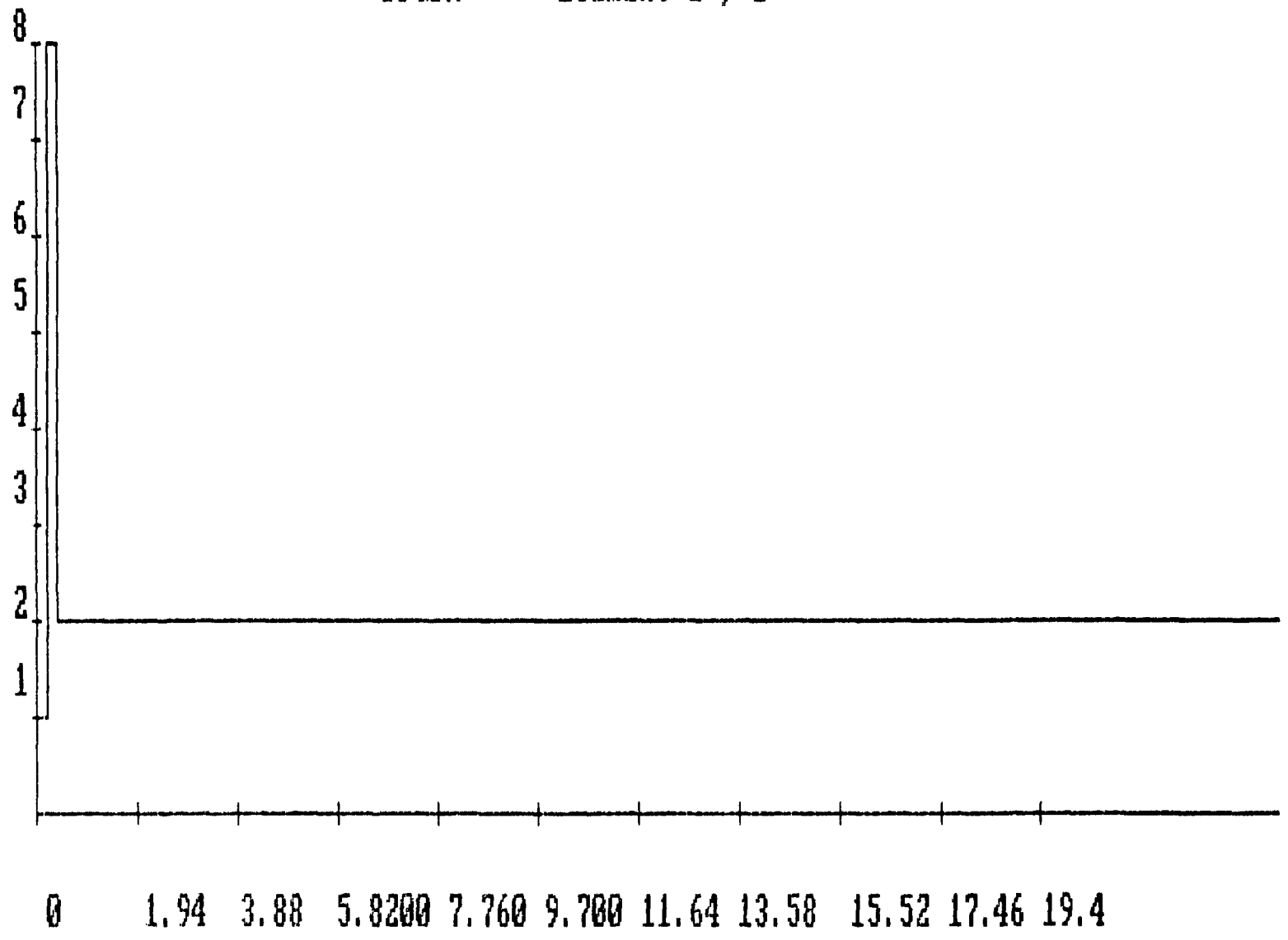

DFGDM Element 1 , 1
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

Figure 3.

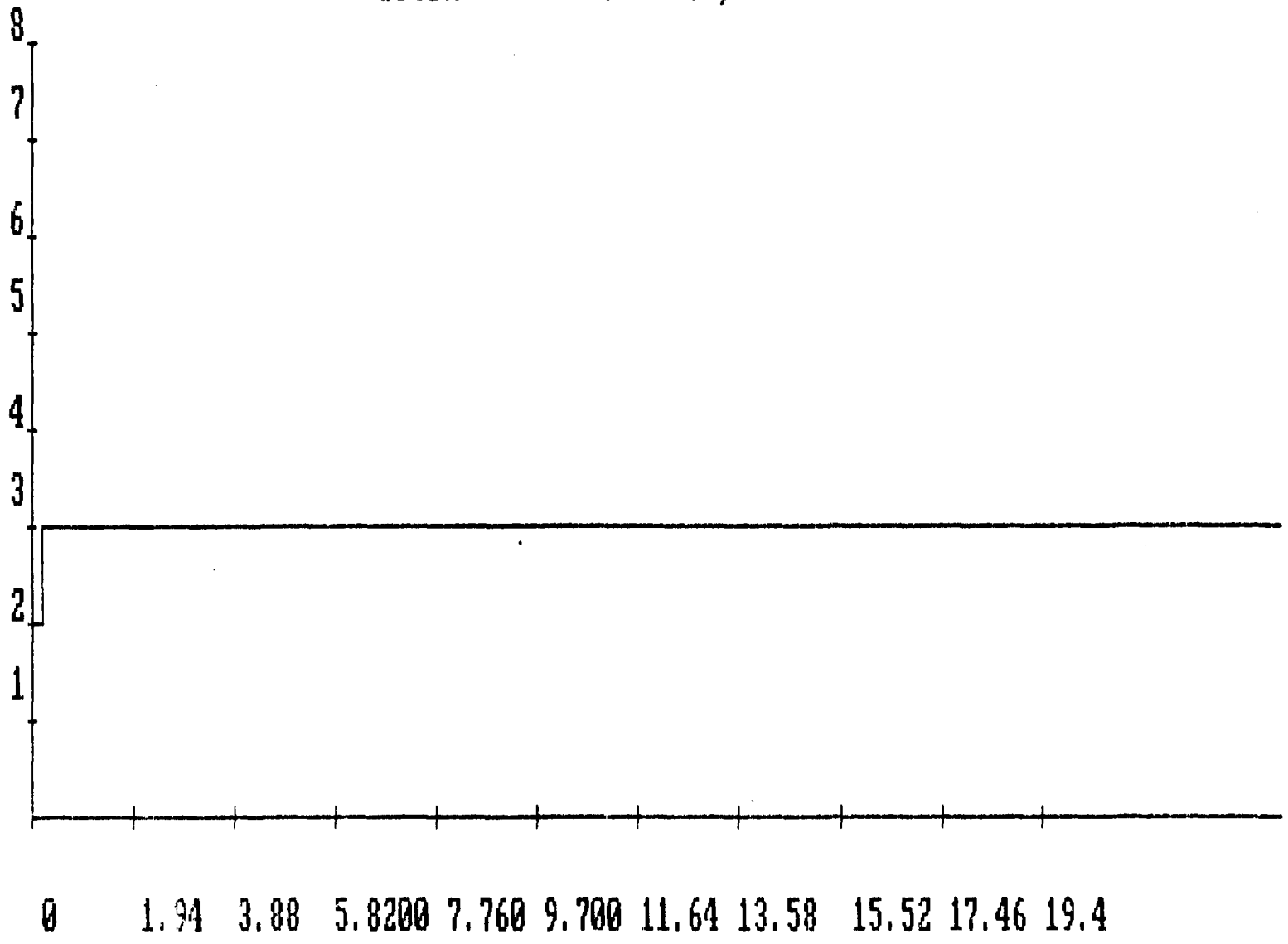

DFGDM Element 1 , 2
8
7
6
5
4
3
2
1
0 1.94 3.88 5.8200 7.760 9.700 11.64 13.58 15.52 17.46 19.4

figure 4

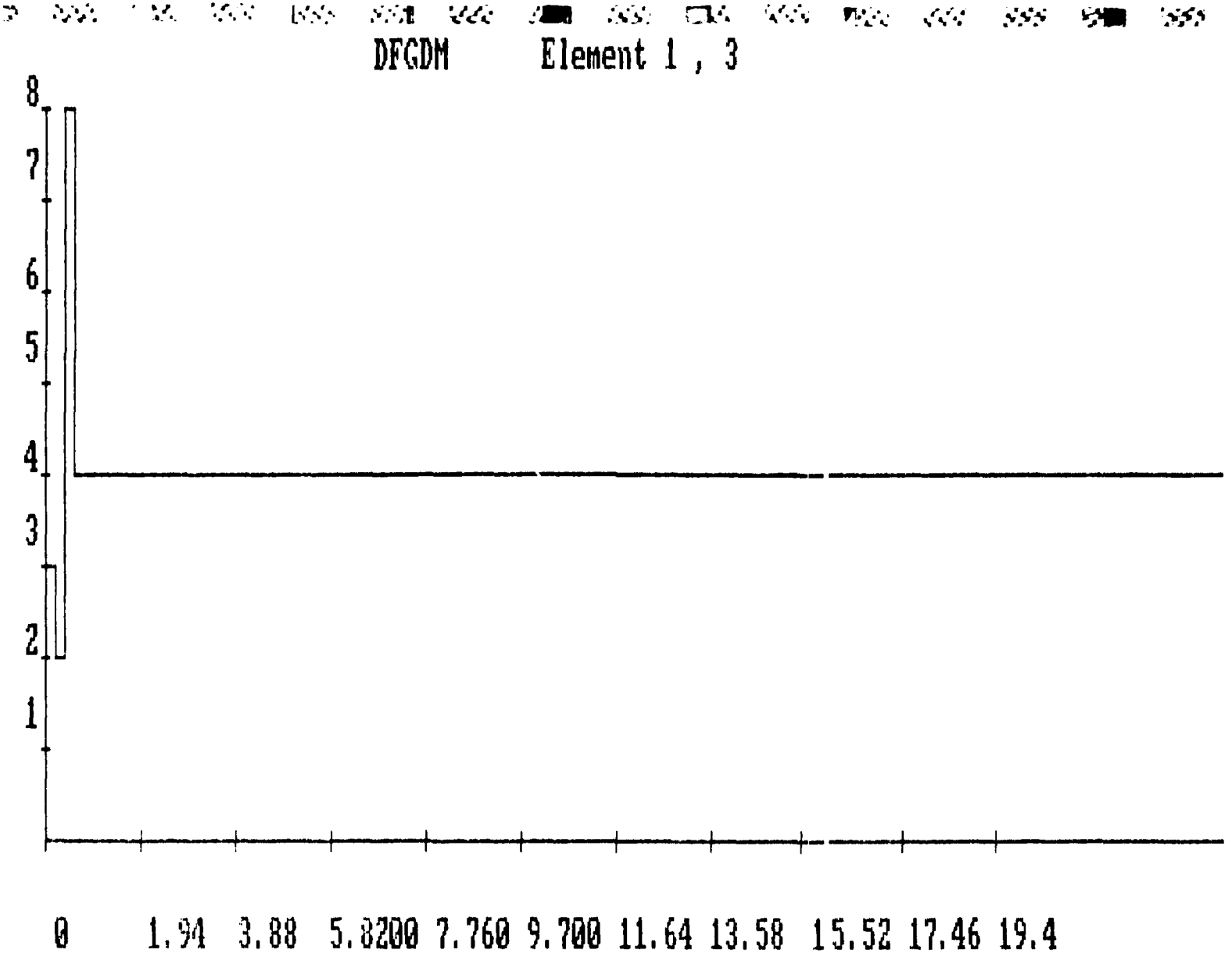

DFGDM Element 1 , 3
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 5

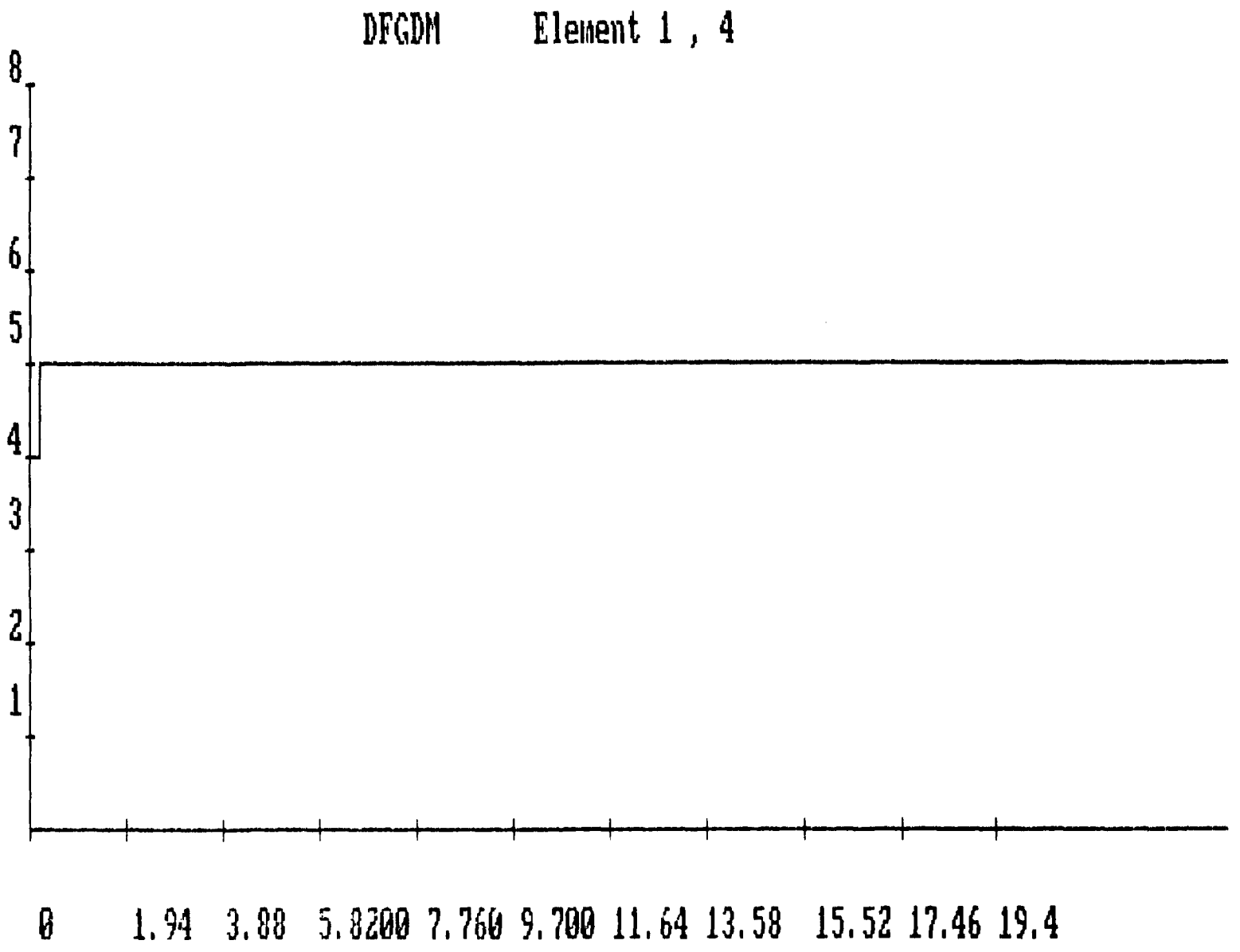

DFGDM Element 1 , 4
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 6

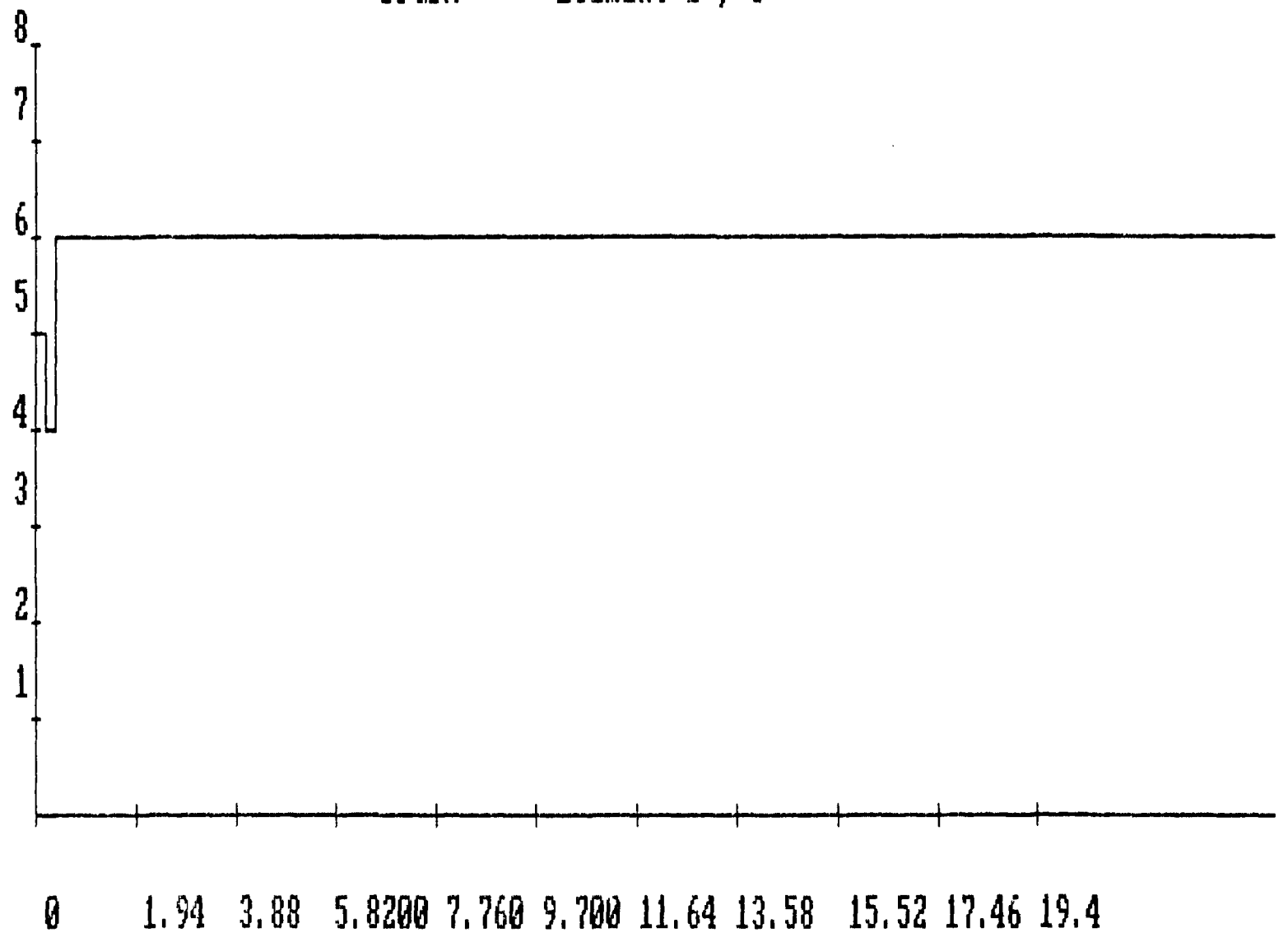

DFGDM Element 1 , 5
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 7.

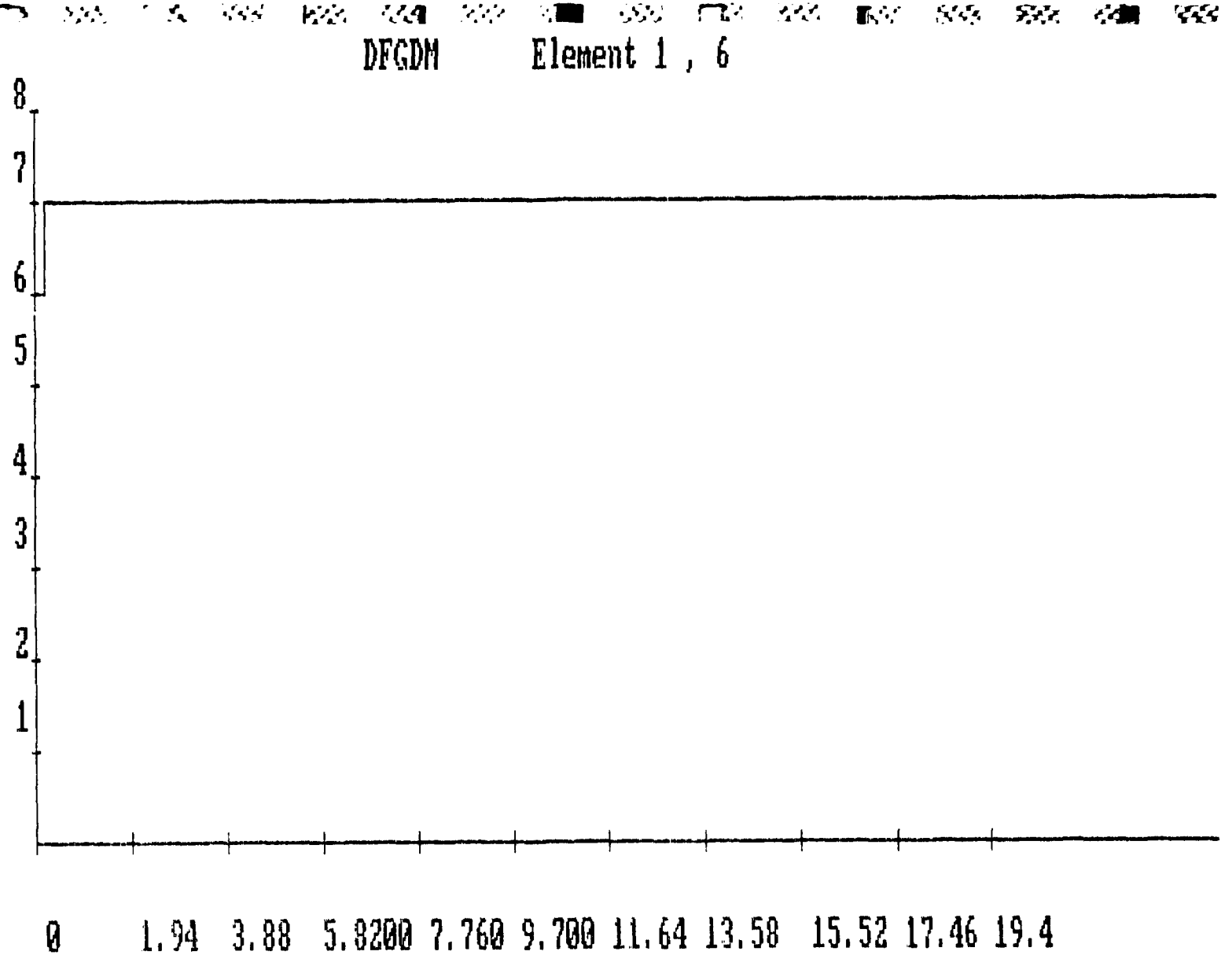

DFGDM Element 1 , 6
8
7
6
5
4
3
2
1
0 1.94 3.88 5.8200 7.760 9.700 11.64 13.58 15.52 17.46 19.4

figure 8

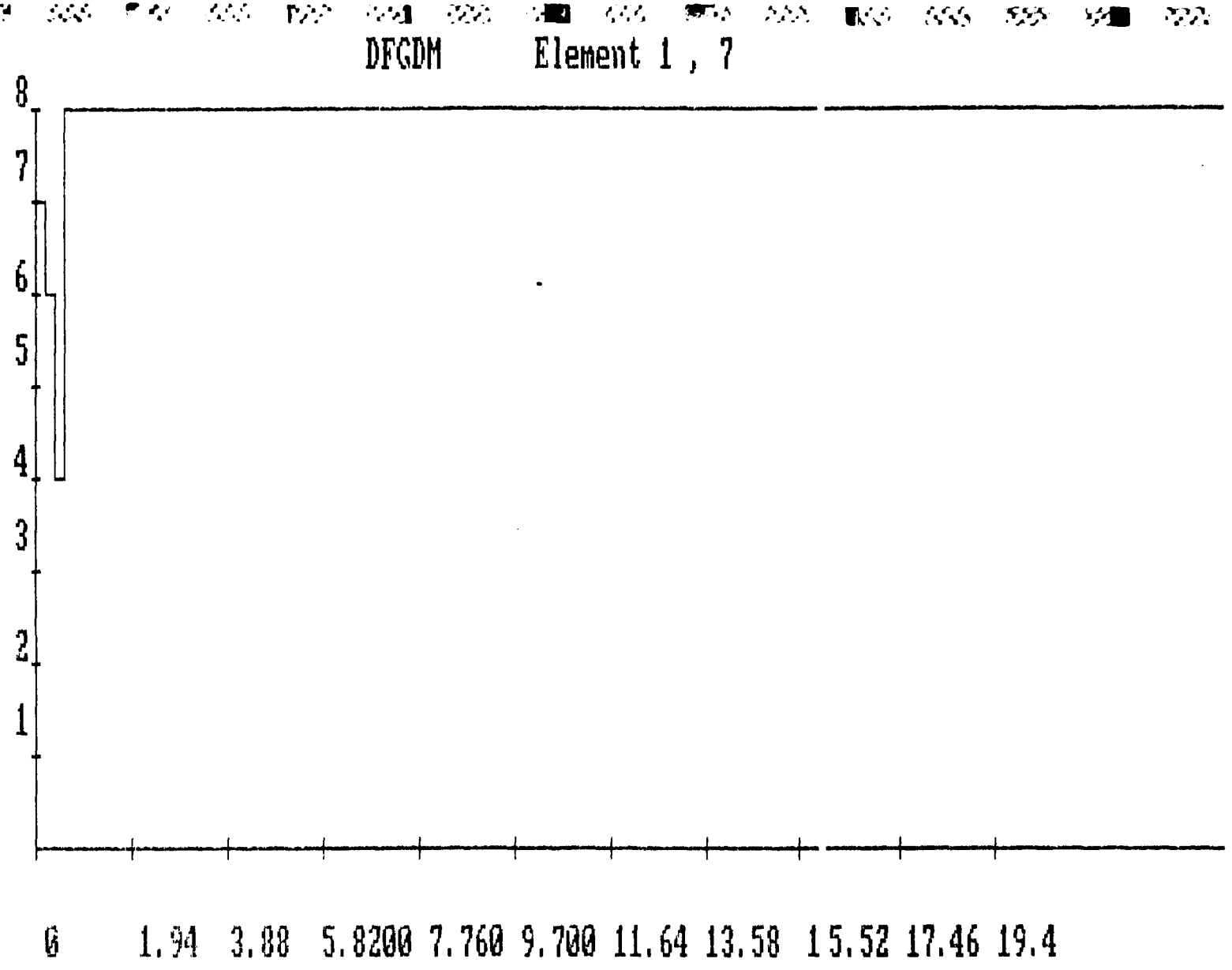

DFGDM Element 1 , 7
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

Figure 9

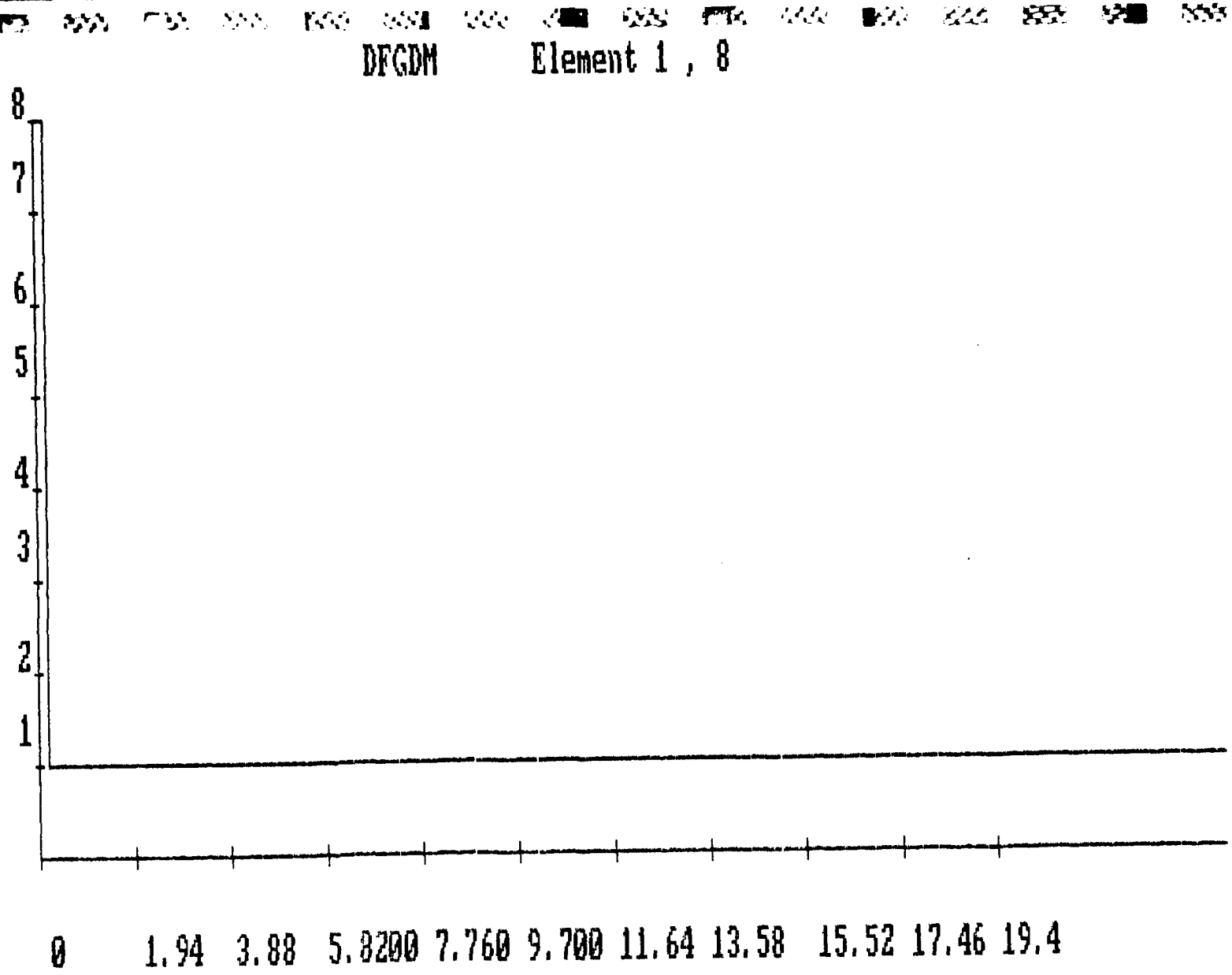

DFGDM Element 1 , 8
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 10

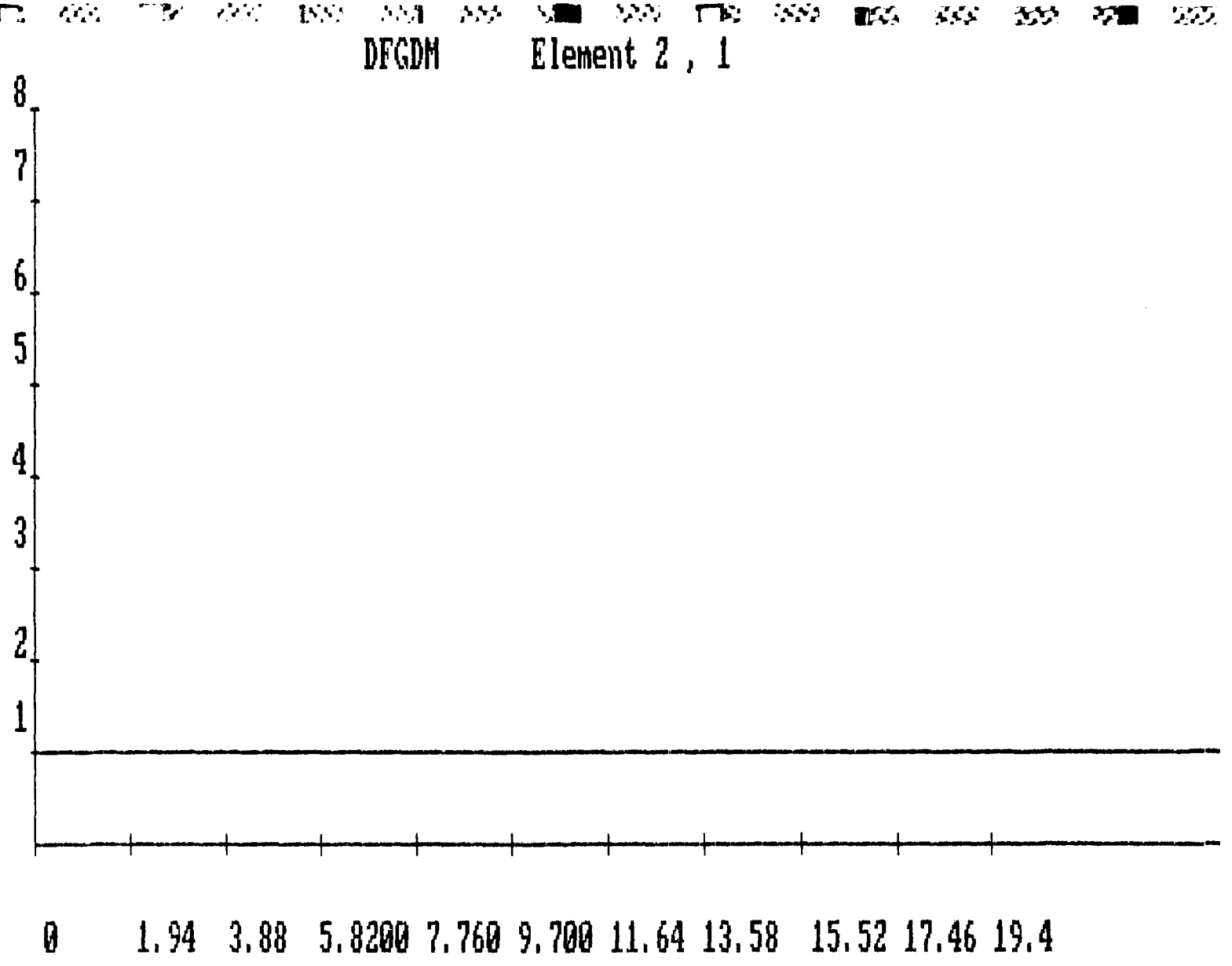

DFGDM Element 2 , 1
8
7
6
5
4
3
2
1
0 1.94 3.88 5.8200 7.760 9.700 11.64 13.58 15.52 17.46 19.4

figure 11.

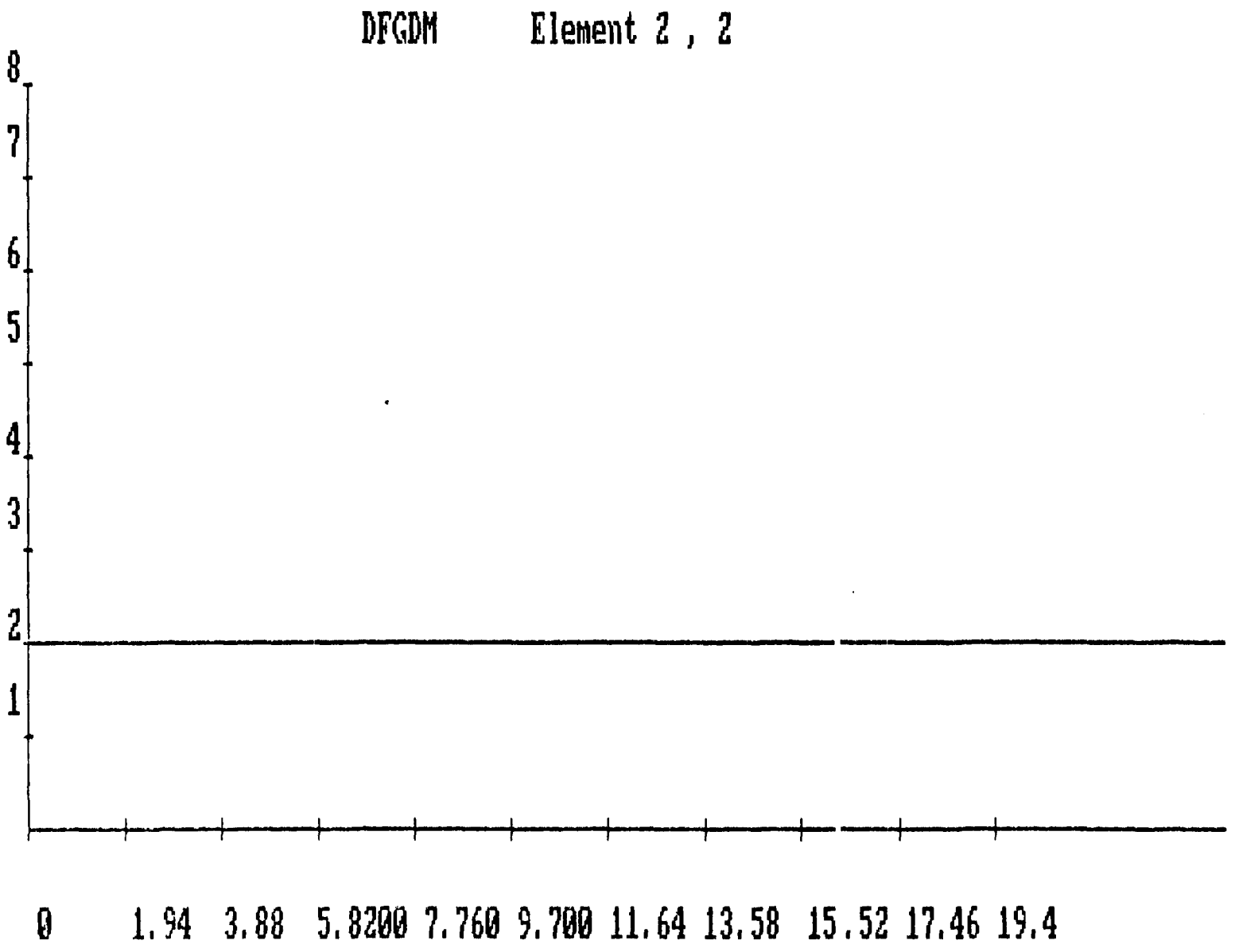

DFGDM Element 2 , 2
8
7
6
5
4
3
2
1
0 1.94 3.88 5.8200 7.760 9.700 11.64 13.58 15.52 17.46 19.4

figure 12

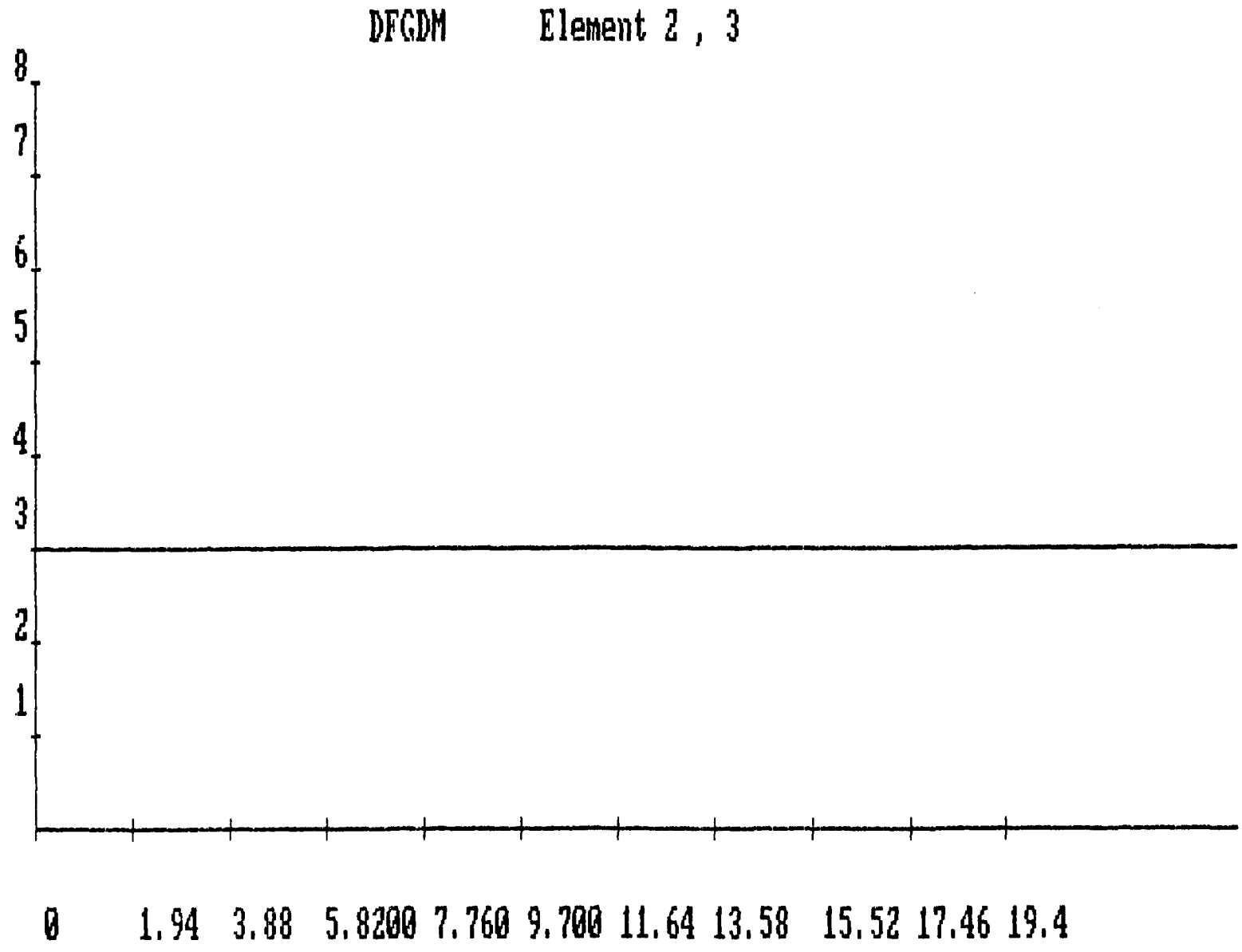

DFGDM Element 2 , 3
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 13

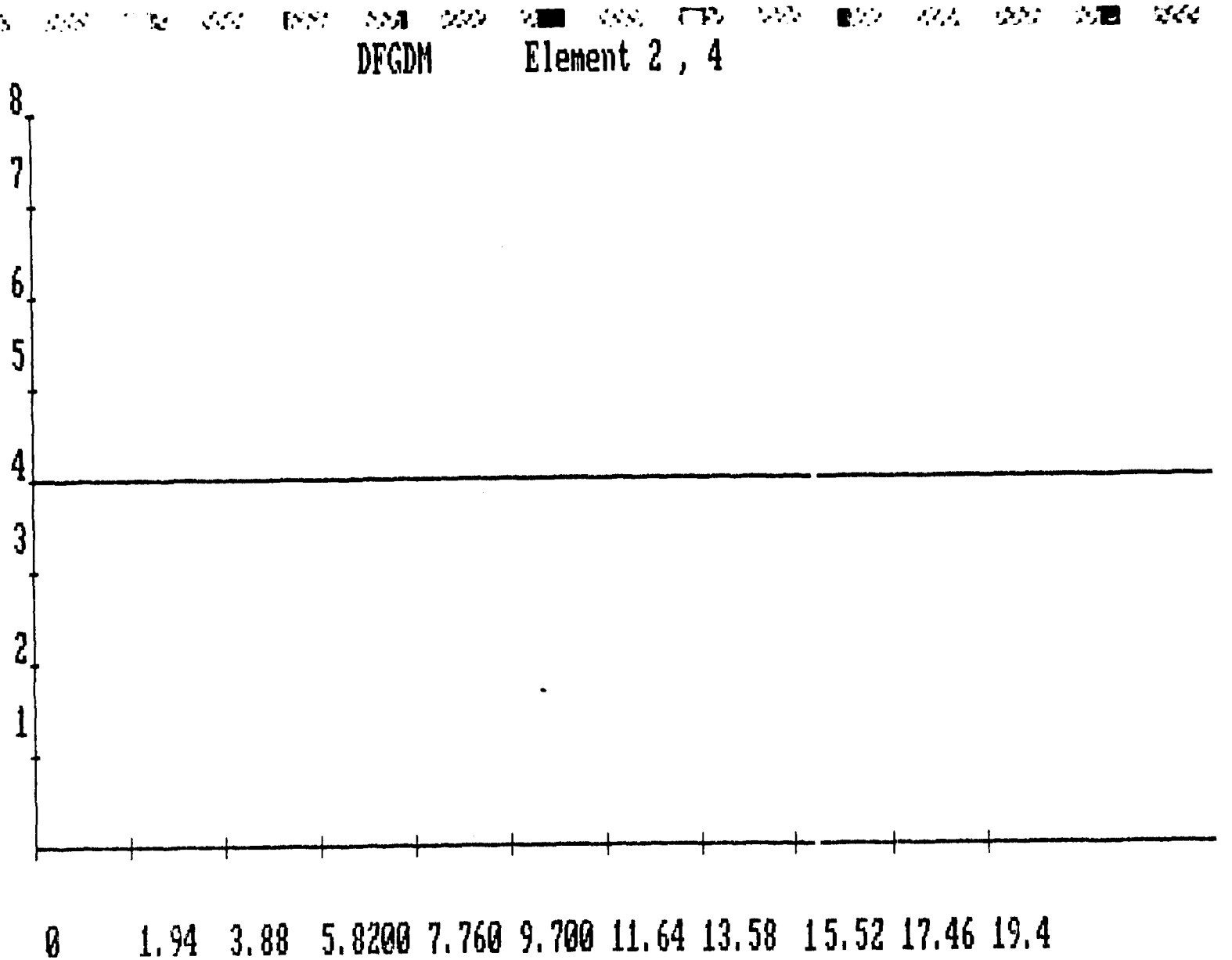

DFGDM Element 2 , 4
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 14

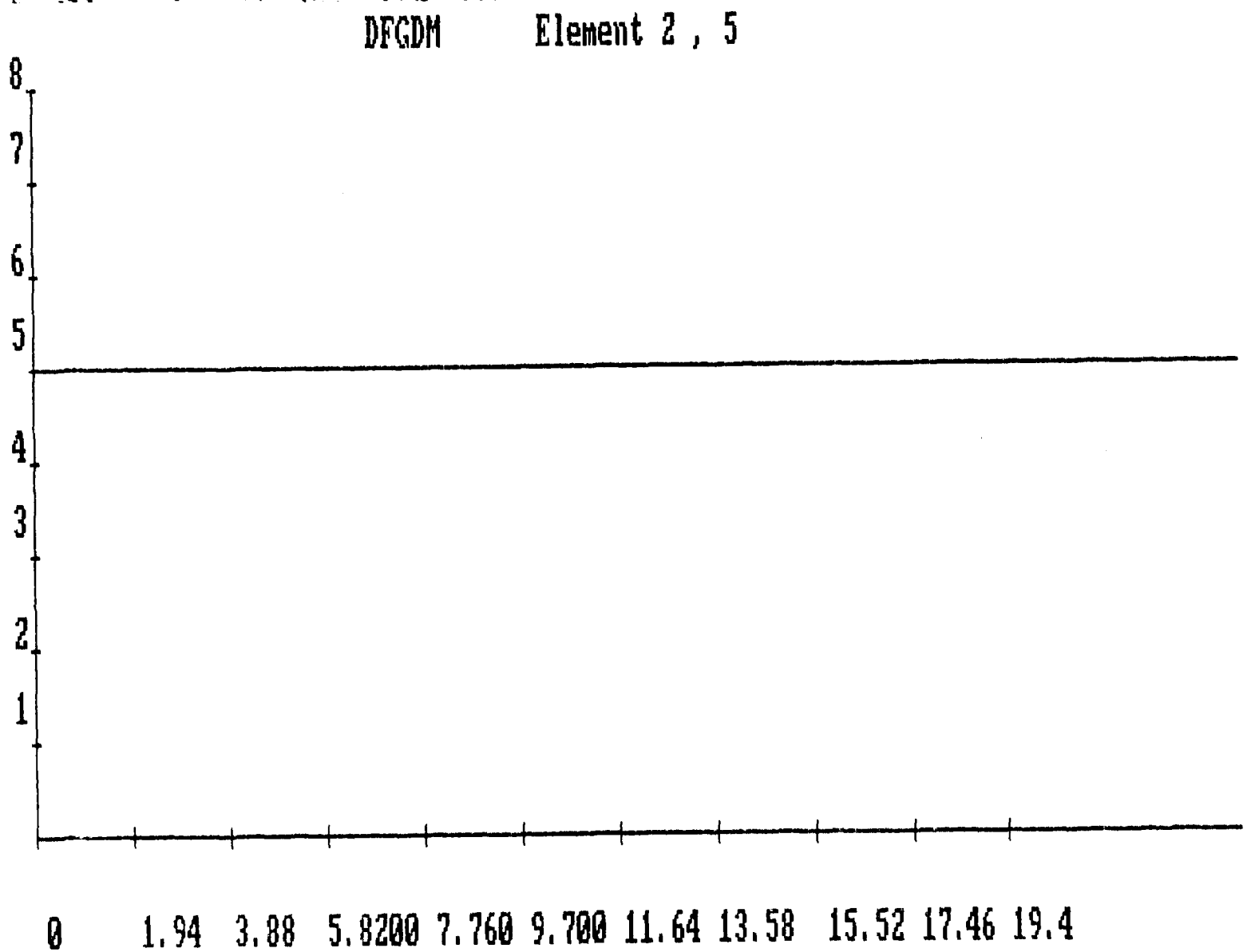

DFGDM Element 2 , 5
8
7
6
5
4
3
2
1
0 1.94 3.88 5.8200 7.760 9.700 11.64 13.58 15.52 17.46 19.4

figure 15.

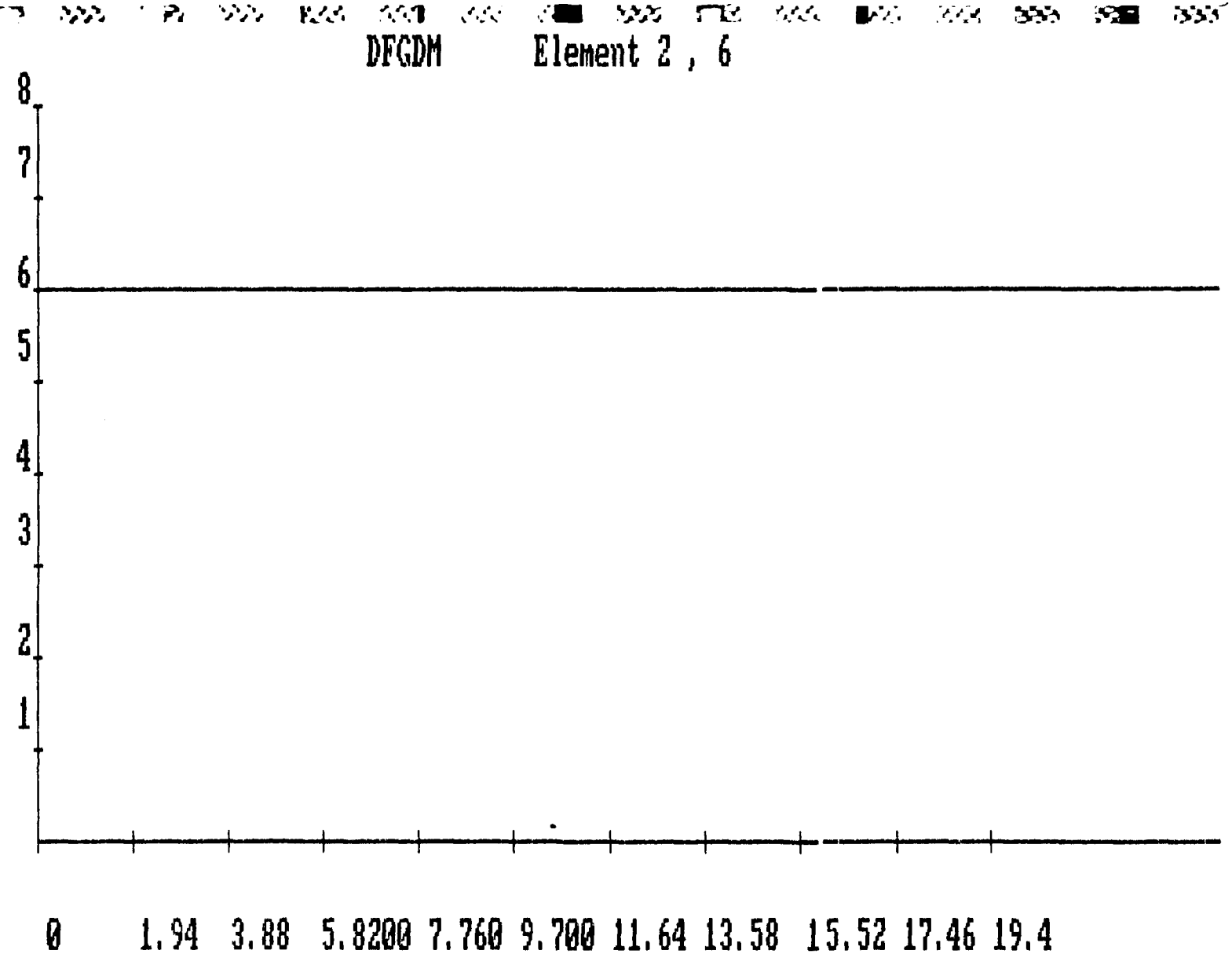

DFGDM Element 2 , 6
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 16.

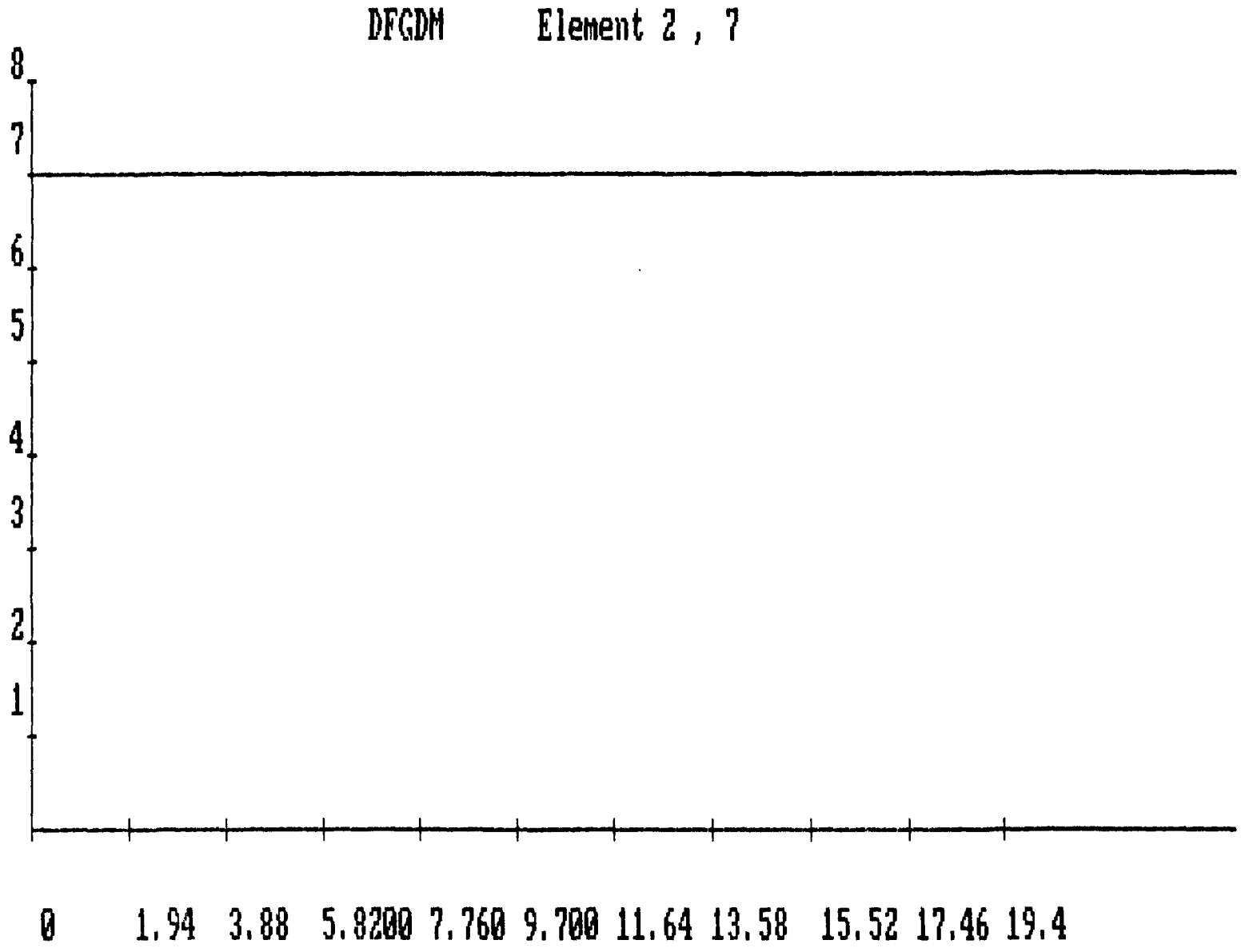

DFGDM Element 2 , 7
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 17.

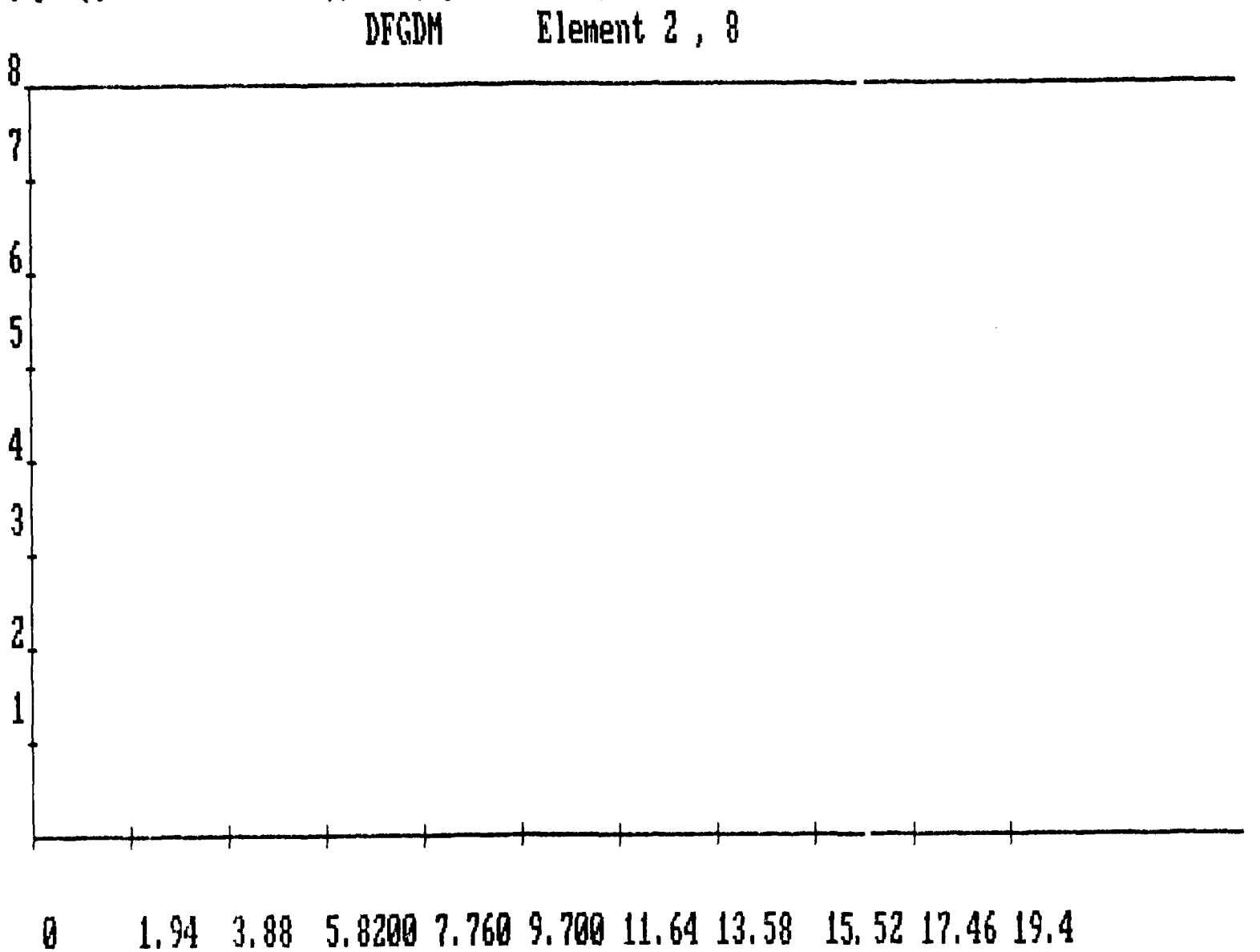

DFGDM Element 2 , 8
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 18.

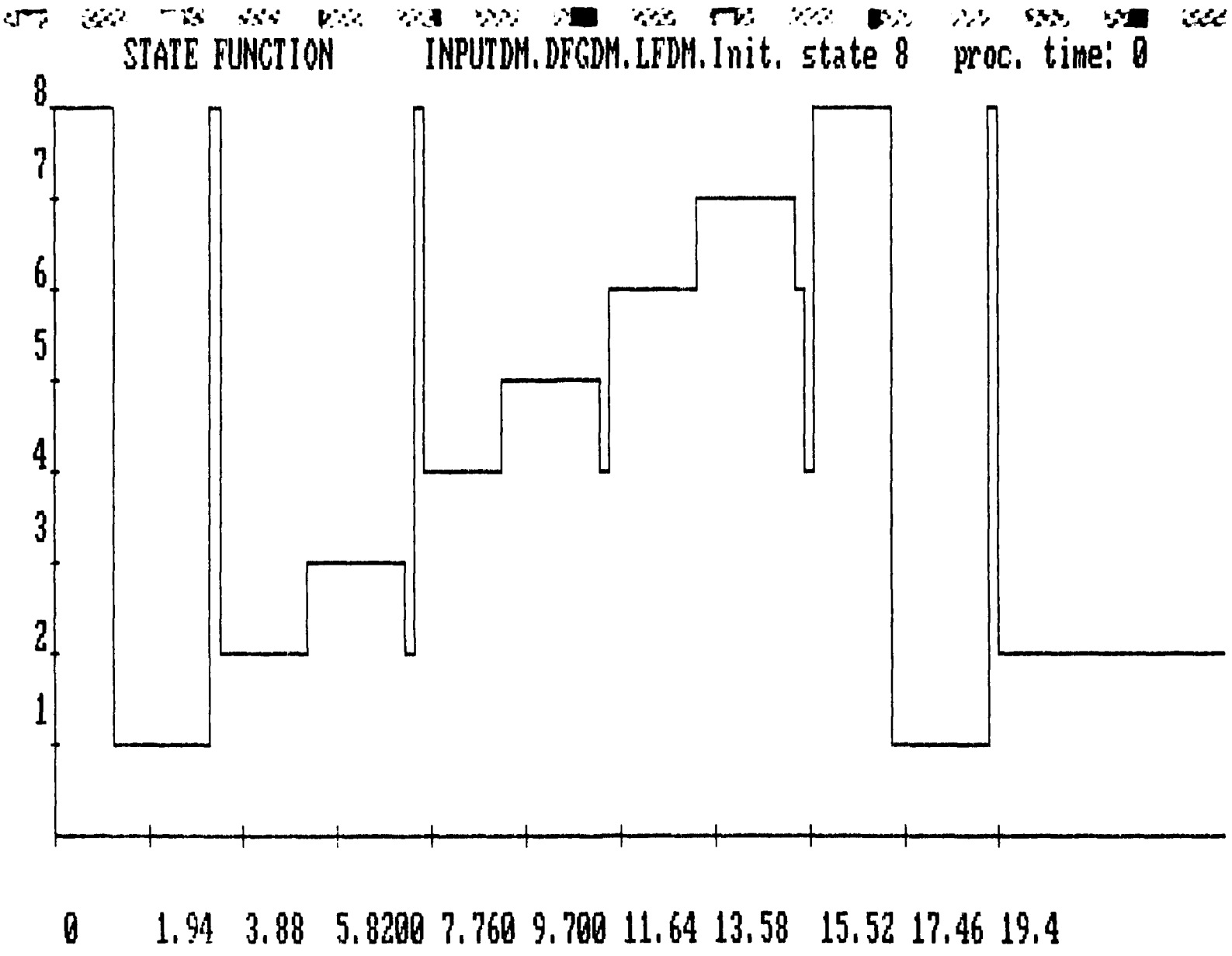

STATE FUNCTION INPUTDM.DFGDM.LFDM.Init. state 8 proc. time: 0
8
7
6
5
4
3
2
1
0
1.94
3.88
5.8200
7.760
9.700
11.64
13.58
15.52
17.46
19.4

figure 19

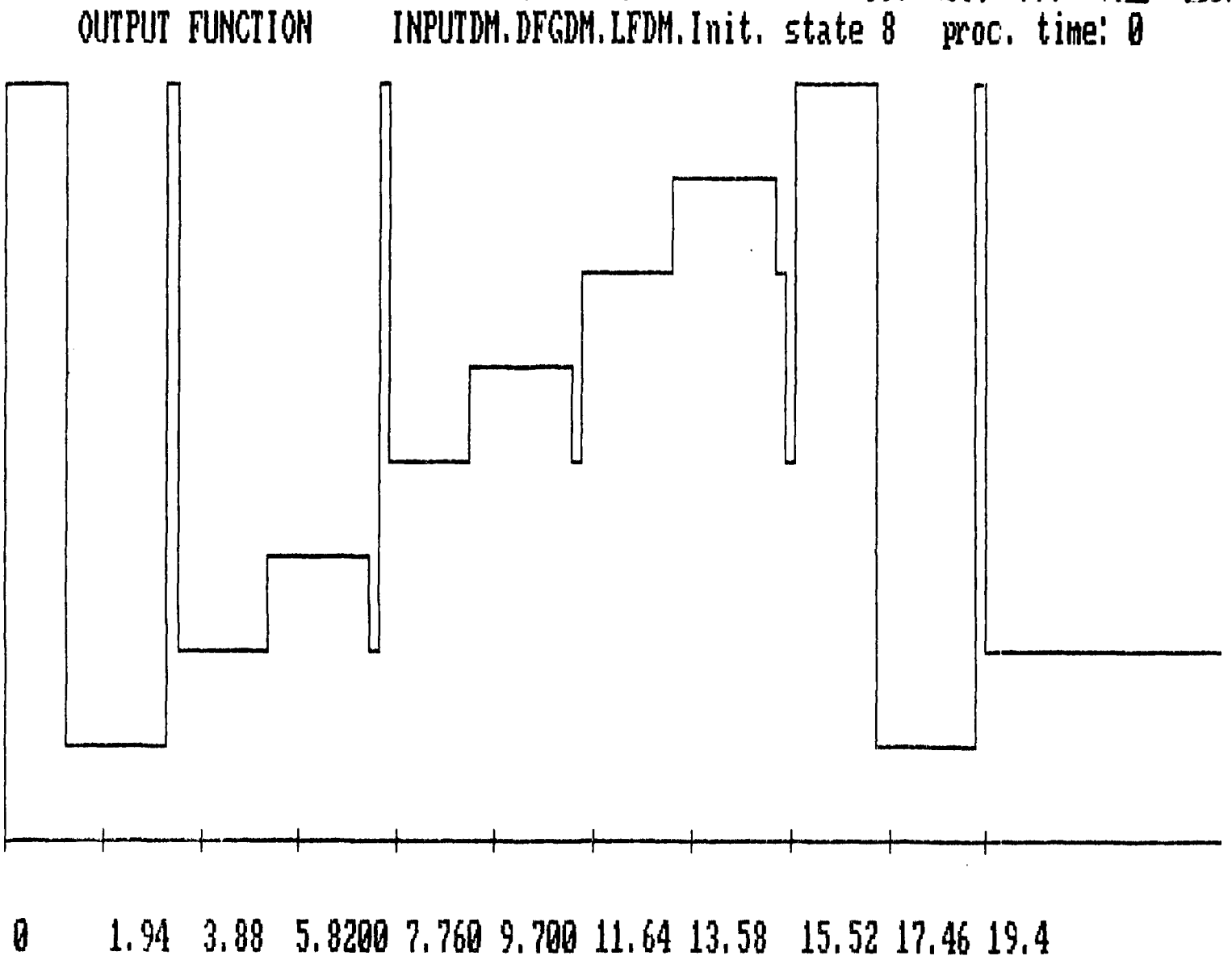

OUTPUT FUNCTION INPUTDM.DFGDM.LFDM.Init. state 8 proc. time: 0
0 1.94 3.88 5.8200 7.760 9.700 11.64 13.58 15.52 17.46 19.4

figure 20

c) Modeling of RCA CDP1854 Programmable UART

MODELING OF RCA CDP1854 PROGRAMMABLE UART

1. INTRODUCTION

An example of the usefulness of the SAM simulator is illustrated here by modeling the behavior of a RCA CDP1854 Programmable Universal Asynchronous Receiver/Transmitter (UART). The 1854 can be programmed to transmit 5-8 bit characters, with or without parity, and one or two stop bits. See attachment for a detailed specification of the 1854 UART. To transmit a character, the character is loaded into the transmitter holding register. The UART then transfers it to the transmitter shift register, where it is serially shifted onto the transmit line.

Due to current limitations in the simulator (only a single input and a single output), and because of the complexity of the 1854, several assumptions are made and limitations are placed on the operation of the 1854.

First, the 1854 expects an external clock input connected to TCLOCK (pin 40) to be 16 times the actual transfer rate of the serial data. To simplify the simulation, TCLOCK is assumed to be identical with the transmit frequency.

Second, the 1854 is assumed to be configured to operate in Mode 0 by grounding pin 2 on the chip. In this mode many important output functions are directly available as output pins on the chip. The chip is also configured to transmit 5 data bits. The specification states that 1.5 stop bits are transmitted after a 5 bit character if the chip is configured for 2 stop bits. The simulation assumes that 2 stop bits are transmitted with 5 bit characters.

2. INPUT FUNCTION

For this simulation, inputs of the 1854 are taken as TCLOCK(H), the external clock input, and THRL(L). Note that the polarity of the signal is designated in parenthesis after the signal. "H" means the signal is activated, or asserted when the signal is high, or logic value 1. "L" means the signal is asserted when the signal is low, or logical value 0. THRL is the transmitter holding register load signal and is active-low.

Because of the limitations of having only a single input, the two binary input signals are encoded into a four level input function. This input function is shown in figure 1. The values of the binary inputs and the level of the input function are related as here:

THRL(L)	TCLOCK(H)	INPUT FUNCTION LEVEL
0	0	4
0	1	1
1	0	2
1	1	3

Again, referring to figure 1, it can be seen that the input signal initially alternates between levels 2 and 3. This is equivalent to the TCLOCK input changing as expected with THRL high, or not asserted.

When the input signal alternates between levels 4 and 1, this is equivalent to the THRL signal being asserted (recall that it is asserted when its value is a logical 0) and the clock input continuing as before. Finally the THRL signal becomes not asserted and the clock continues as before.

What the input function is modeling is the normal operation of the 1854 UART and an asynchronous signal which loads a character into the holding register of the UART. The time axis for all plots is expressed in units of microseconds. Thus, a time interval of 2 is equal to 2 microseconds and a time interval of .3 is equal to 300 nanoseconds.

3. DIGITAL FUNCTION GENERATORS

Before explaining the DFGs themselves, it is first necessary to explain the meaning of each of the 9 state values. Five bit characters are being transmitted along with a start bit and two stop bits. In addition, the UART may be in a mark state (no data available to transmit). This implies the following 9 states along with their state levels:

State Level	State Description
1	start bit transmission
2	data bit 0 transmission
3	data bit 1 transmission
4	data bit 2 transmission
5	data bit 3 transmission
6	data bit 4 transmission
7	first stop bit transmission
8	second stop bit transmission
9	mark transmission

Some of the DFGs for the simulation are shown in figures 2-19. Only the DFGs for input levels 1 and 3 are shown since the DFGs for input levels 2 and 4 are constant. This makes sense because during levels 2 and 4, the clock input,

TCLOCK, is low and it is assumed that transitions occur on the rising edge of the clock pulse.

To explain the DFGs in figures 2-19, see figure 2 as a typical example. Continuity of state is assumed in the DFG and after a time interval of .3 [300 ns] the state transitions from state 1 to state 2. The 300 ns delay reflects the specification for the 1854.

Note in figure 18 that after 300 ns, the state transitions from state 8 to state 1. In other words, the UART transmits another start bit after transmitting the last stop bit. This is true assuming another character is available to transmit after the current character has been sent.

4. STATE FUNCTION

The state function versus time is shown in figure 20 assuming an initial state of 9 [mark state]. On the rising edge of the next clock pulse, the UART transitions to state 1, then 2, and so on. This shows that the 1854 is transmitting a character, bit by bit, as it proceeds from one state to the next.

In actual operation of the 1854, it would not proceed from a mark state to start bit transmission state unless the THRL input signal is pulsed to load a character. With the limited number of states in this simulation, it is not possible to know if the transmitter holding register is empty unless the THRE signal is monitored as another input. This deviation from actual operation in the simulation does not obviate the results, however.

5. OUTPUT FUNCTION

Three output signals are modeled for the 1854. SDO[H], the serial data output of the UART is where the output data is observed. THRE[H], transmitter holding register empty, is asserted when the contents of the transmitter holding register have been transferred to the transmitter shift register. TSRE[H], transmitter shift register empty, is asserted when the last stop bit of the character has been transmitted and there is no character in the holding register.

THRE and TSRE become not asserted when the input signal THRL is asserted. After THRL becomes not asserted, the character that has been loaded into the holding register is transferred to the shift register and THRE is asserted.

The three binary output signals are encoded into a single 8 level output function as in figure 21. The output signals and the corresponding output function level are shown below:

TSRE(H)	THRE(H)	SDO(H)	Output Function Level
0	0	0	8
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

The output function waveform assumes that a five bit character "00101" is transmitted, with bit 0 , a logical 1, transmitted first. Referring to figure 21, the output function starts at level 7 (initially, holding register and shift register are empty, a mark or logical 1 is output on SDO). After the next clock pulse the output level drops to 2 (holding register still empty, start bit transmitted). The output momentarily shifts to level 3 then to level 1 indicating that bit 0 is being transmitted (logical 1), and meanwhile, the THRL input signal is asserted to clear THRE.

Next, bit 1 is transmitted (logical 0) and THRE is re-asserted because the character has been transferred to the shift register and the holding register is now empty. Then bits 2 through 8 are transmitted, and two stop bits. Assuming the holding register has been refilled, the transmitter continues to transmit, beginning with a start bit, and so on.

6. SUMMARY

This 1854 UART example is a good illustration of the utility of the SAM simulator. It shows the basic operation of the UART for transmitting 5 bit characters. This example also surfaces several problems in the simulator.

First, it is difficult to "see" the transitions on the individual binary inputs and outputs since they are encoded in a single multi-level signal. A more useful simulator would allow separate inputs to be specified individually. The same statement is true for the outputs.

Second, it is difficult to accurately model what happens when the last stop bit of a character is transmitted. The UART has to sample the THRE signal. If this signal is asserted, then enter the mark state. If this signal is not asserted, transfer a character from the holding register to

the shift register, enter the transmit start bit state, and assert THRE. The difficulty is that THRE needs to be used as both an input and output signal. The current simulator does not allow outputs to be fed back to inputs.

A less desirable way to circumvent this problem is to add 9 more states, defined as for the first 9 states except that one state means THRE is asserted and the other state means THRE is not asserted. This immediately doubles the requirments of the model, however.

Further work will allow the simulator to better model the precise behavior of asynchronous devices, and thus provide more accurate results.

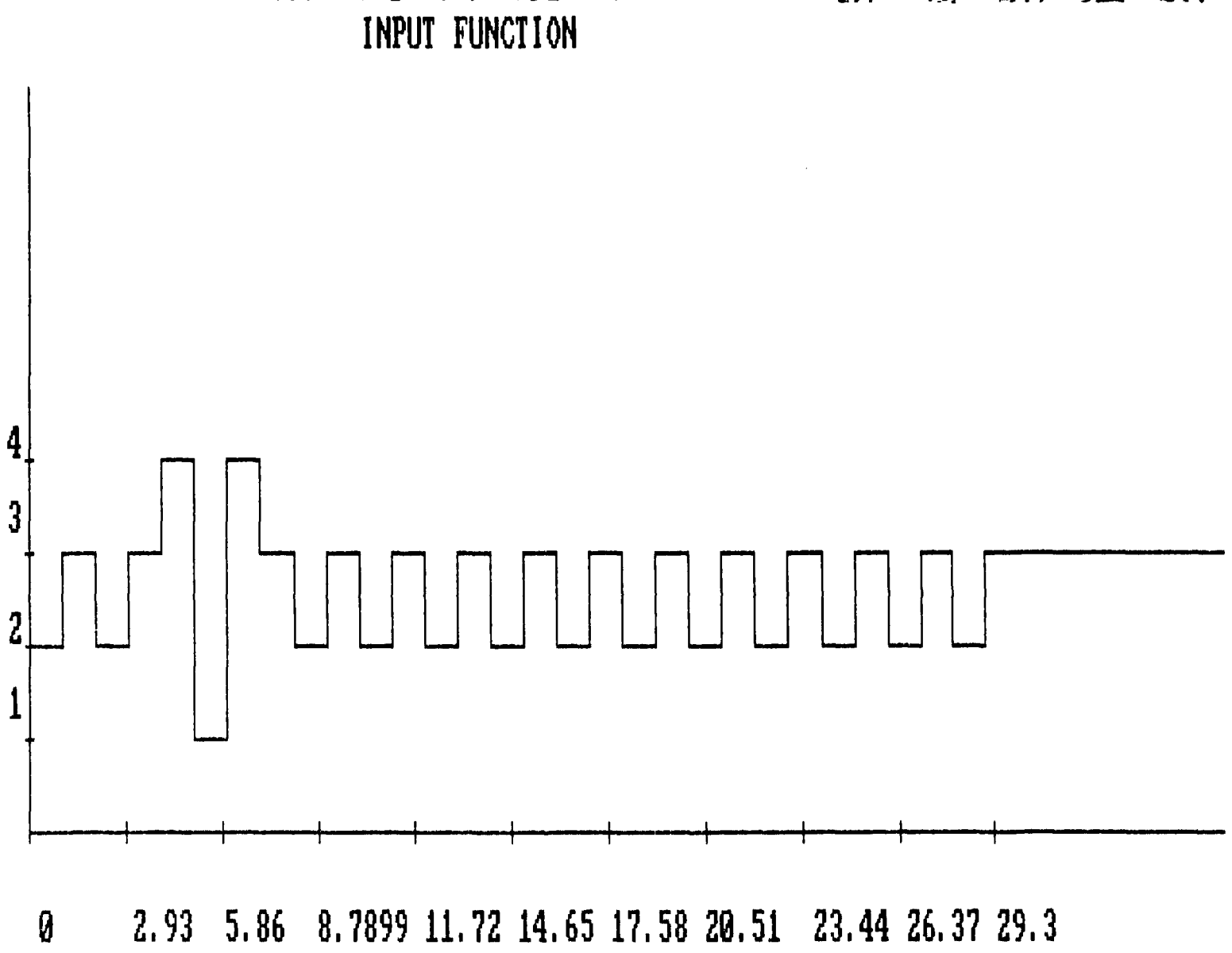

INPUT FUNCTION
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

Figure 1.

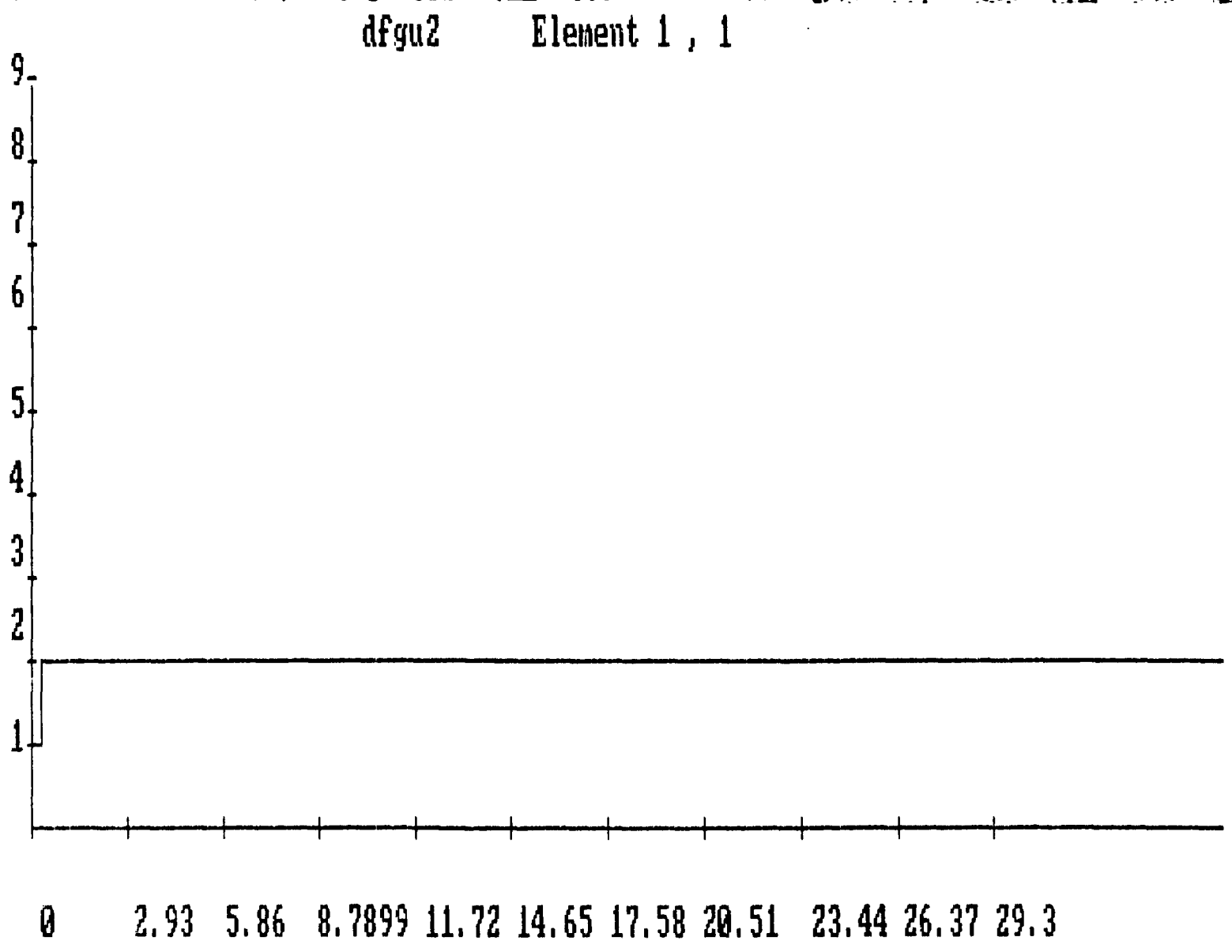

dfgu2 Element 1 , 1
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 2.

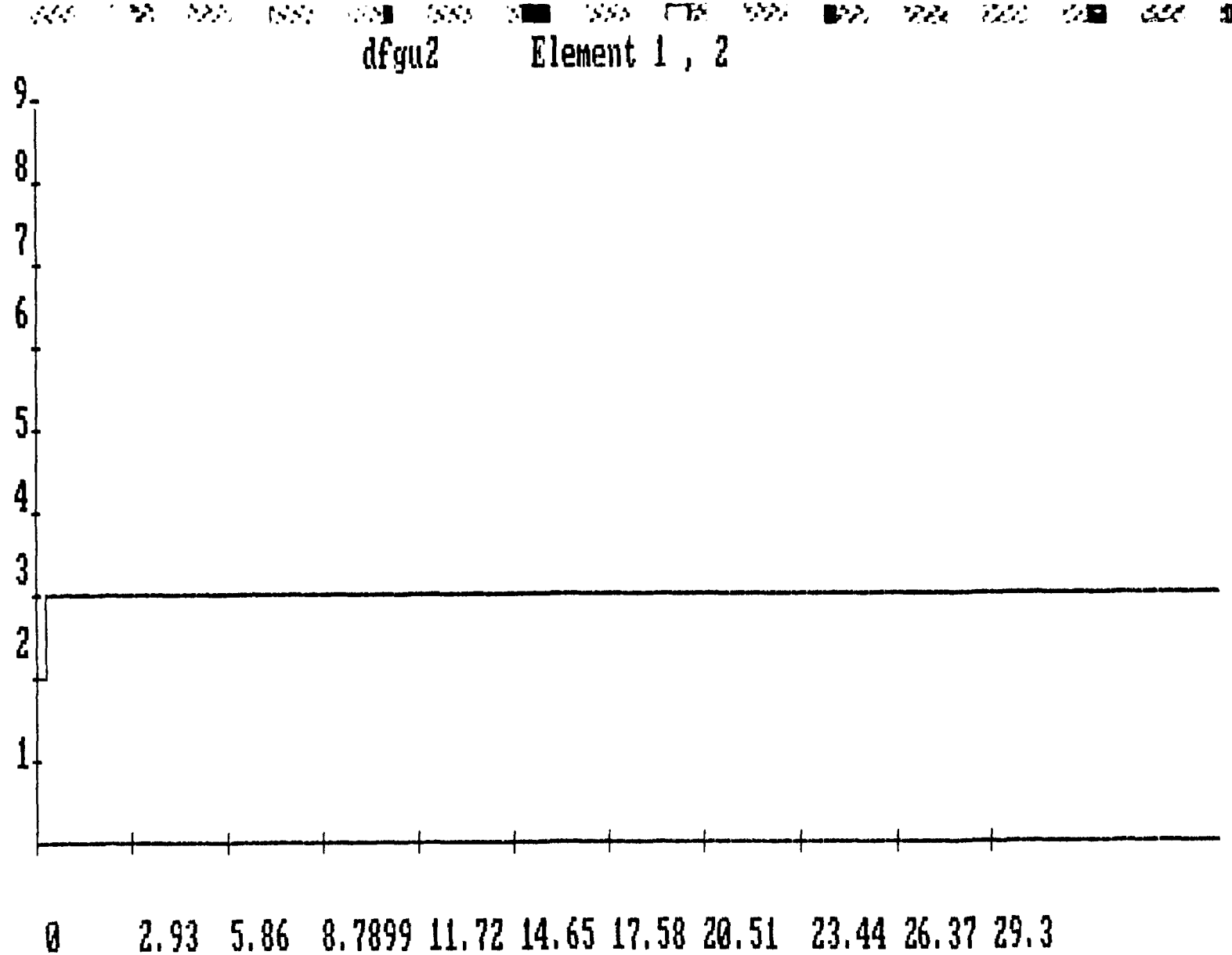

dfgu2 Element 1 , 2
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 3.

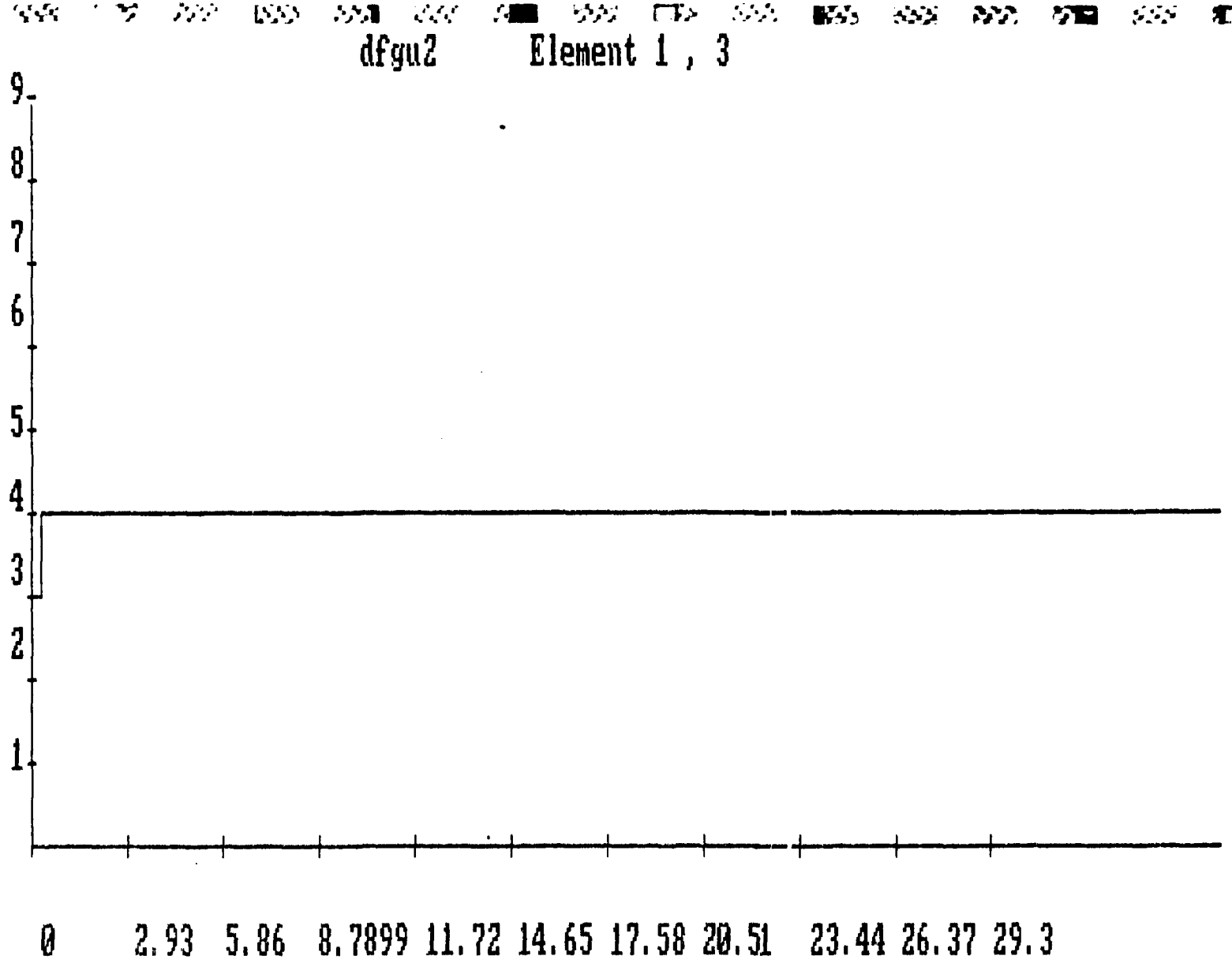

dfgu2 Element 1 , 3
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 4.

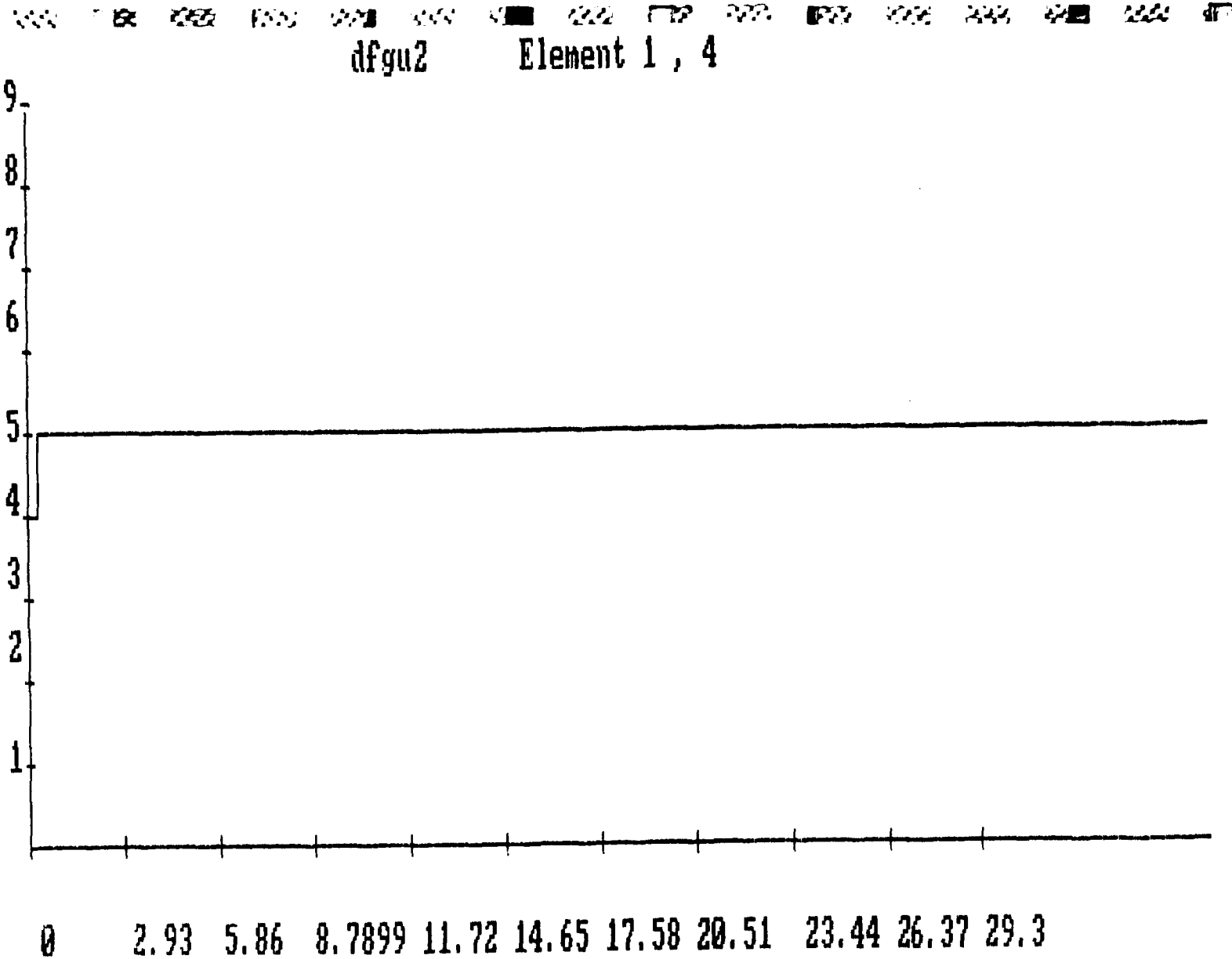

dfgu2 Element 1 , 4
9
8
7
6
5
4
3
2
1
0 2.93 5.86 8.7899 11.72 14.65 17.58 20.51 23.44 26.37 29.3

figure 5.

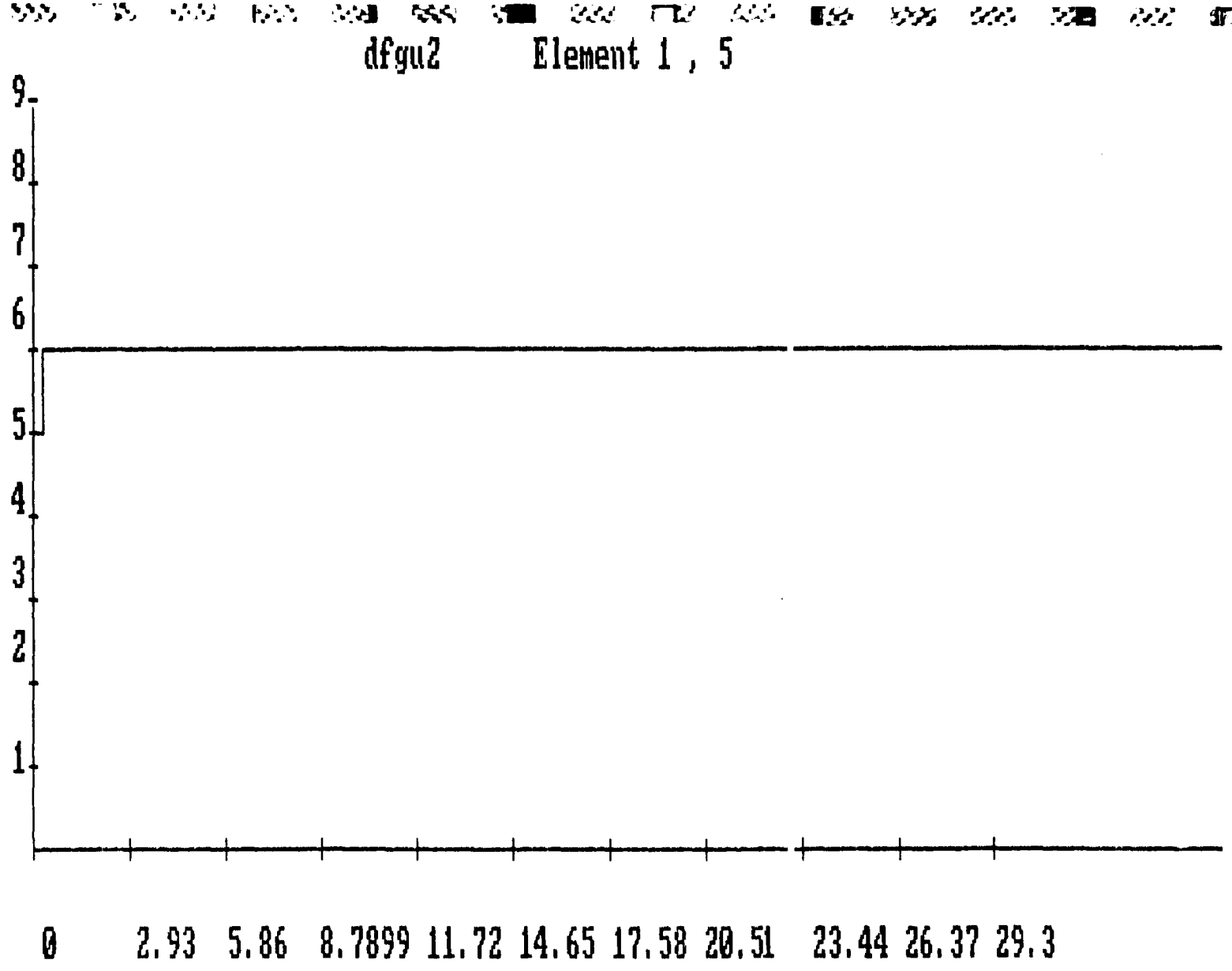

dfgu2 Element 1 , 5
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

Figure 6.

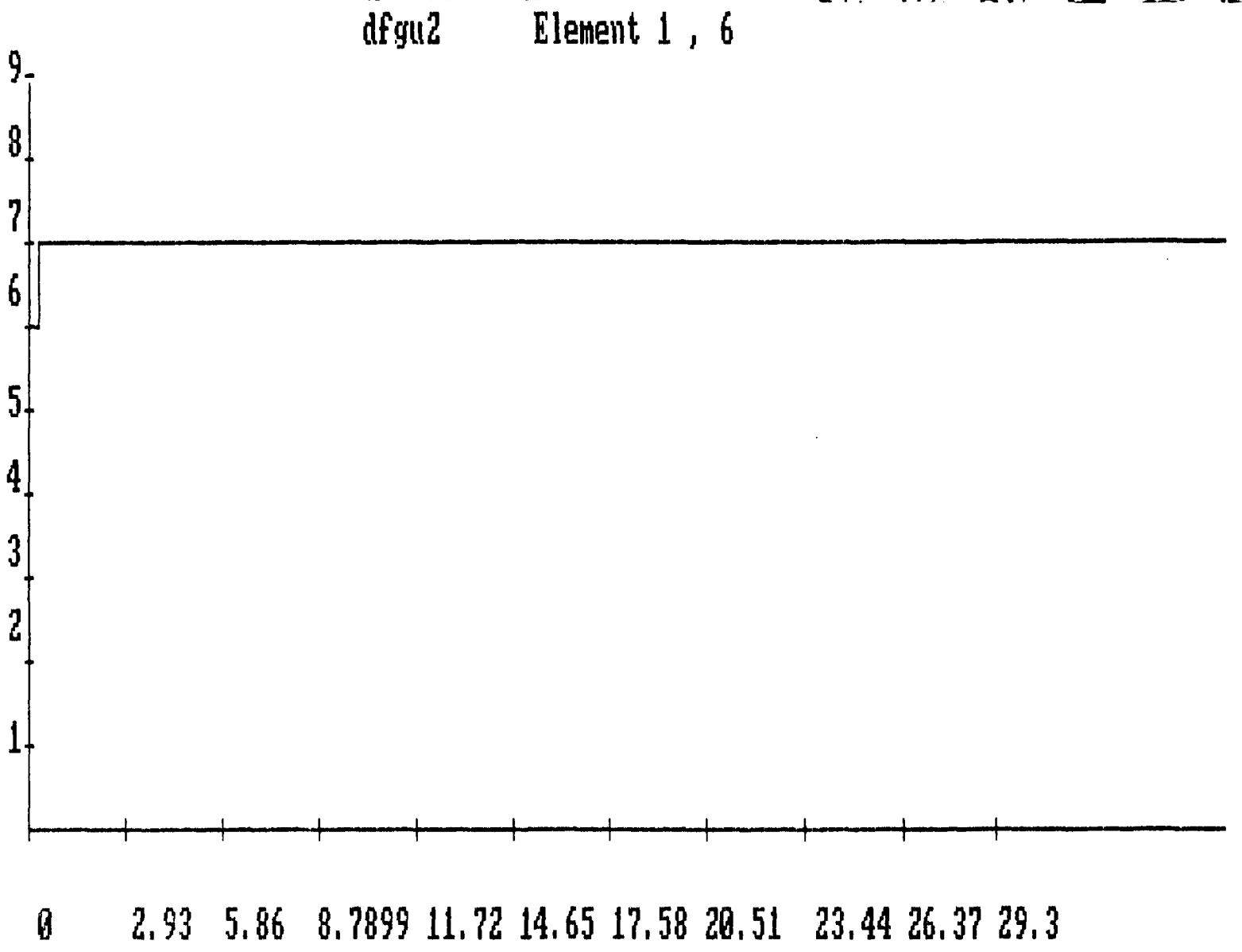

dfgu2 Element 1 , 6
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 7.

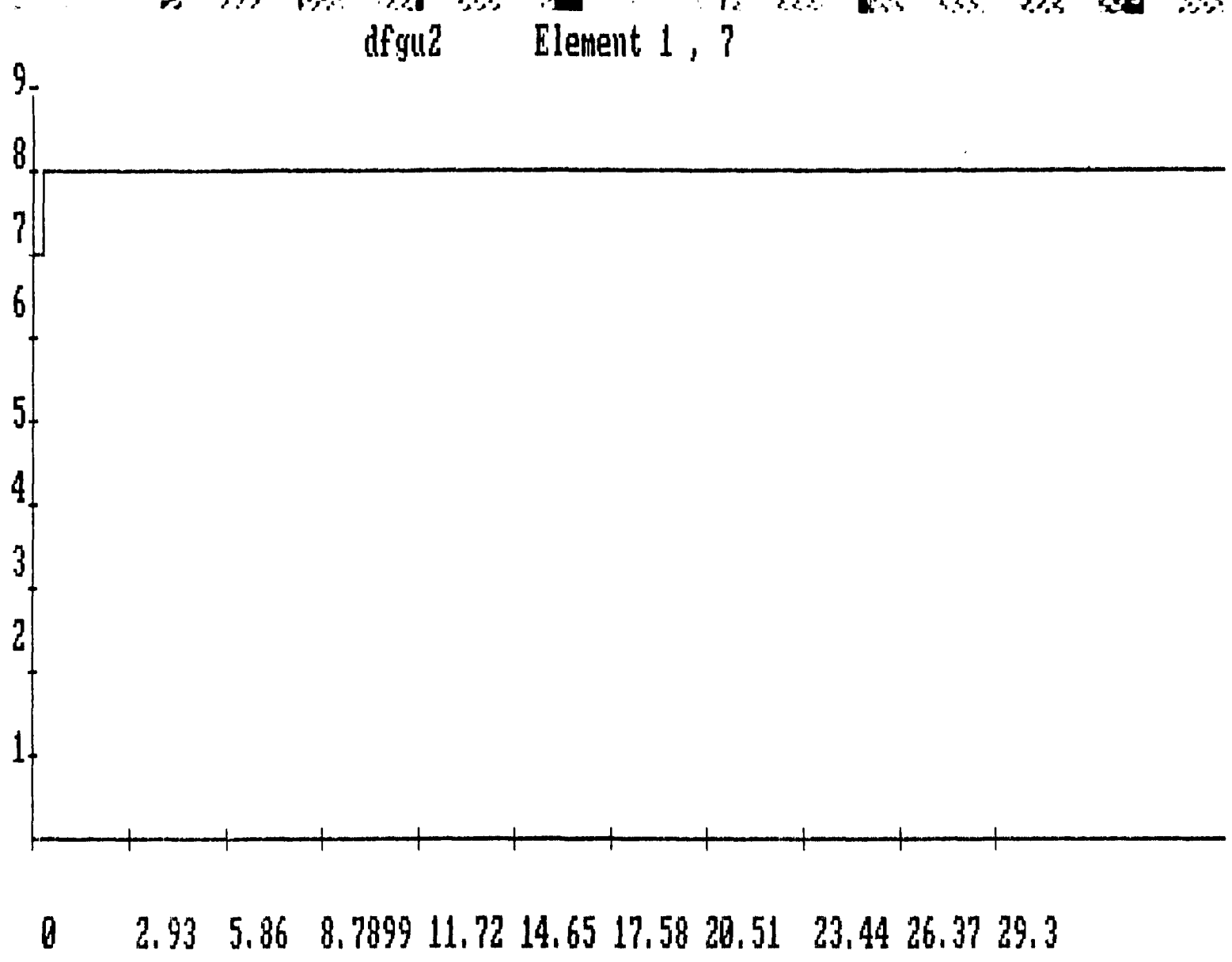

dfgu2 Element 1 , 7
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 8.

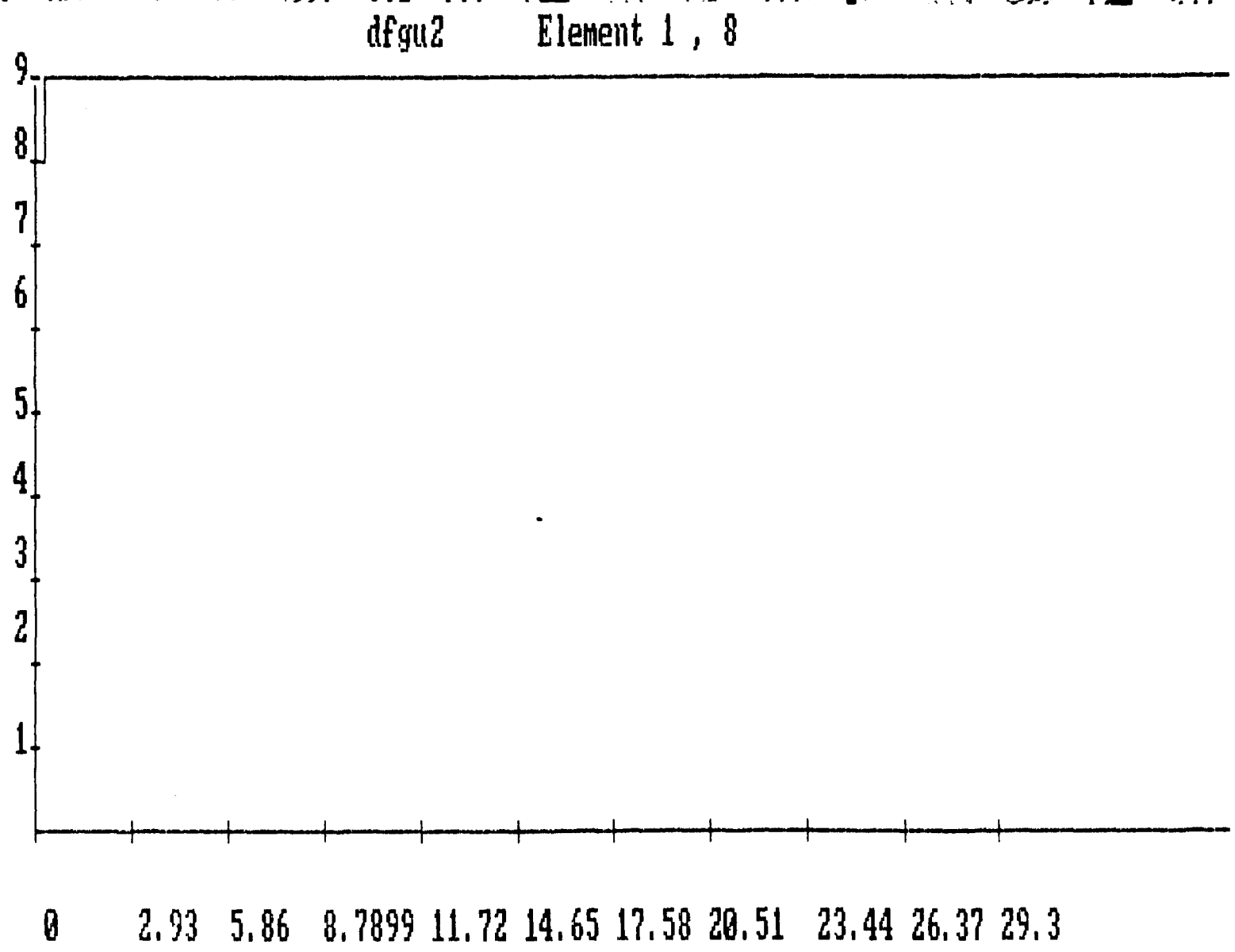

dfgu2 Element 1 , 8
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 9.

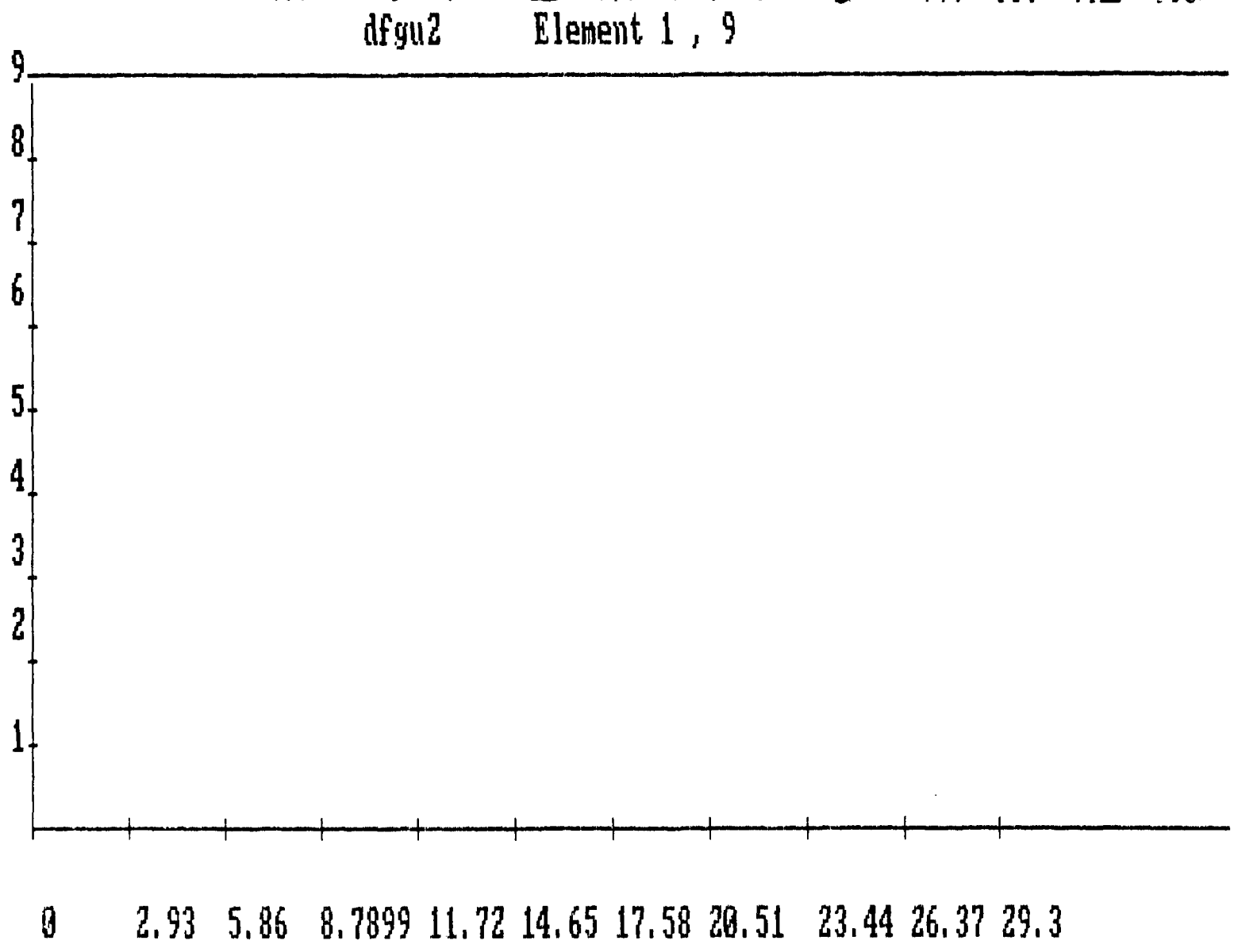

dfgu2 Element 1 , 9
9
8
7
6
5
4
3
2
1
0 2.93 5.86 8.7899 11.72 14.65 17.58 20.51 23.44 26.37 29.3

figure 10.

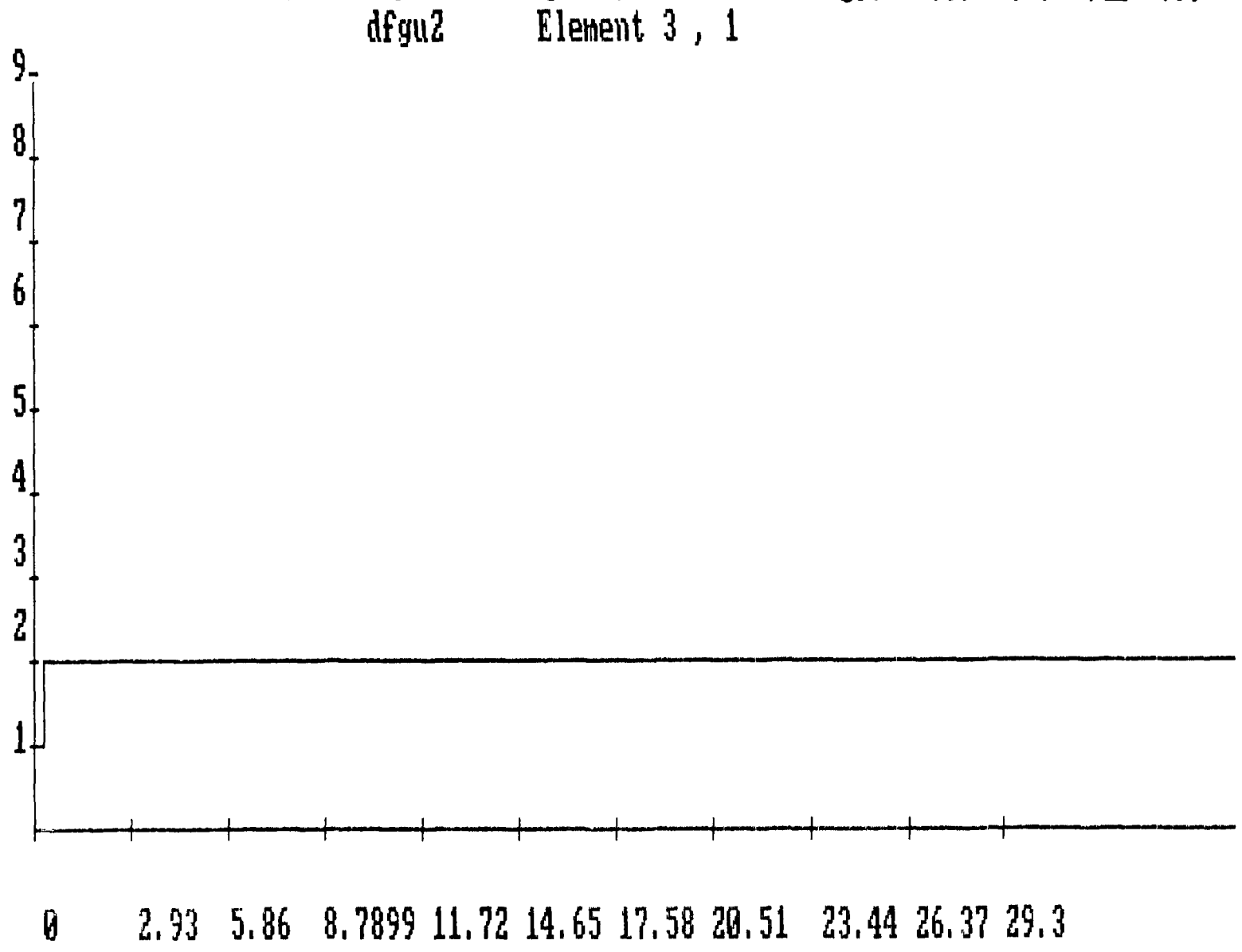

dfgu2 Element 3 , 1
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 11.

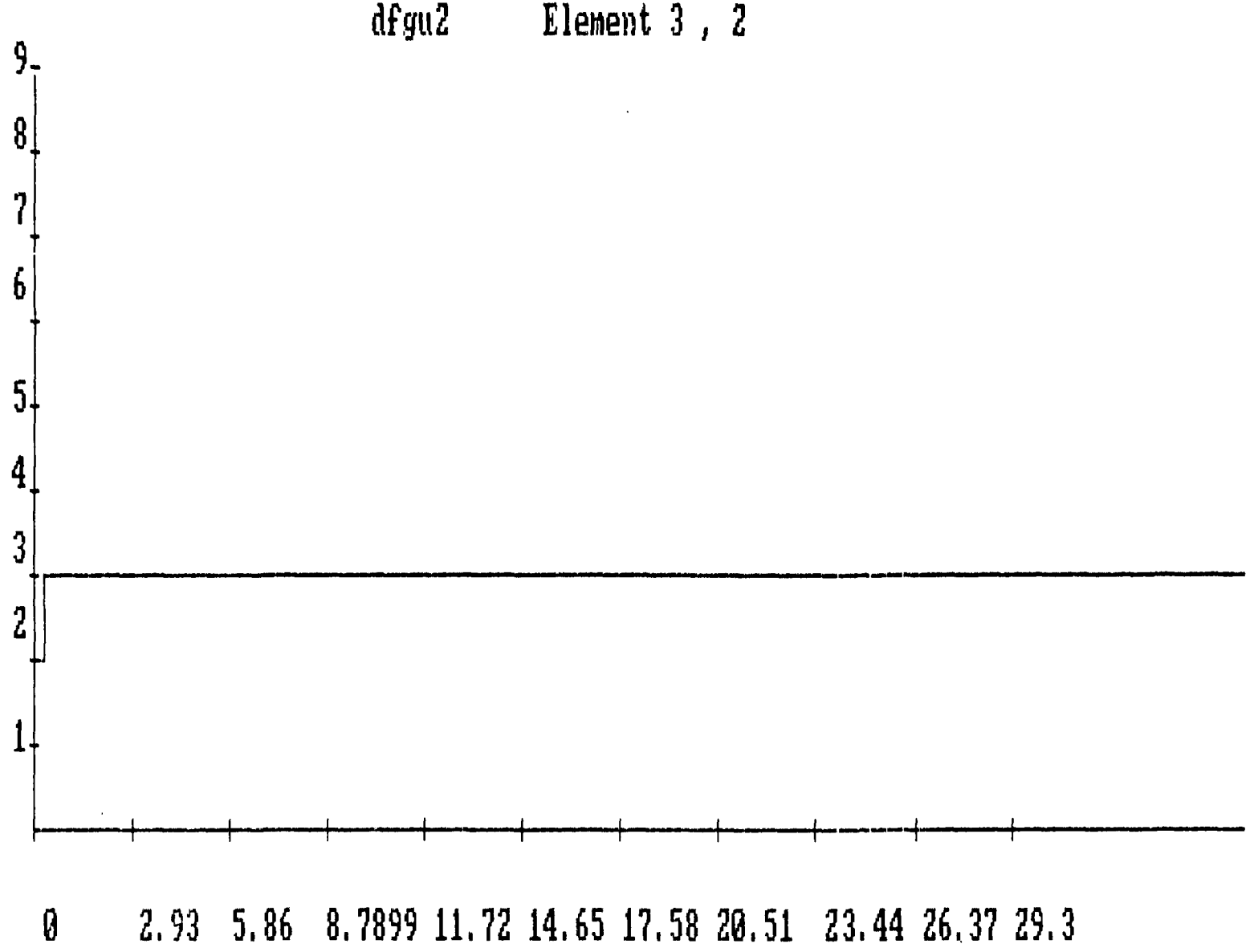

dfgu2 Element 3 , 2
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 12.

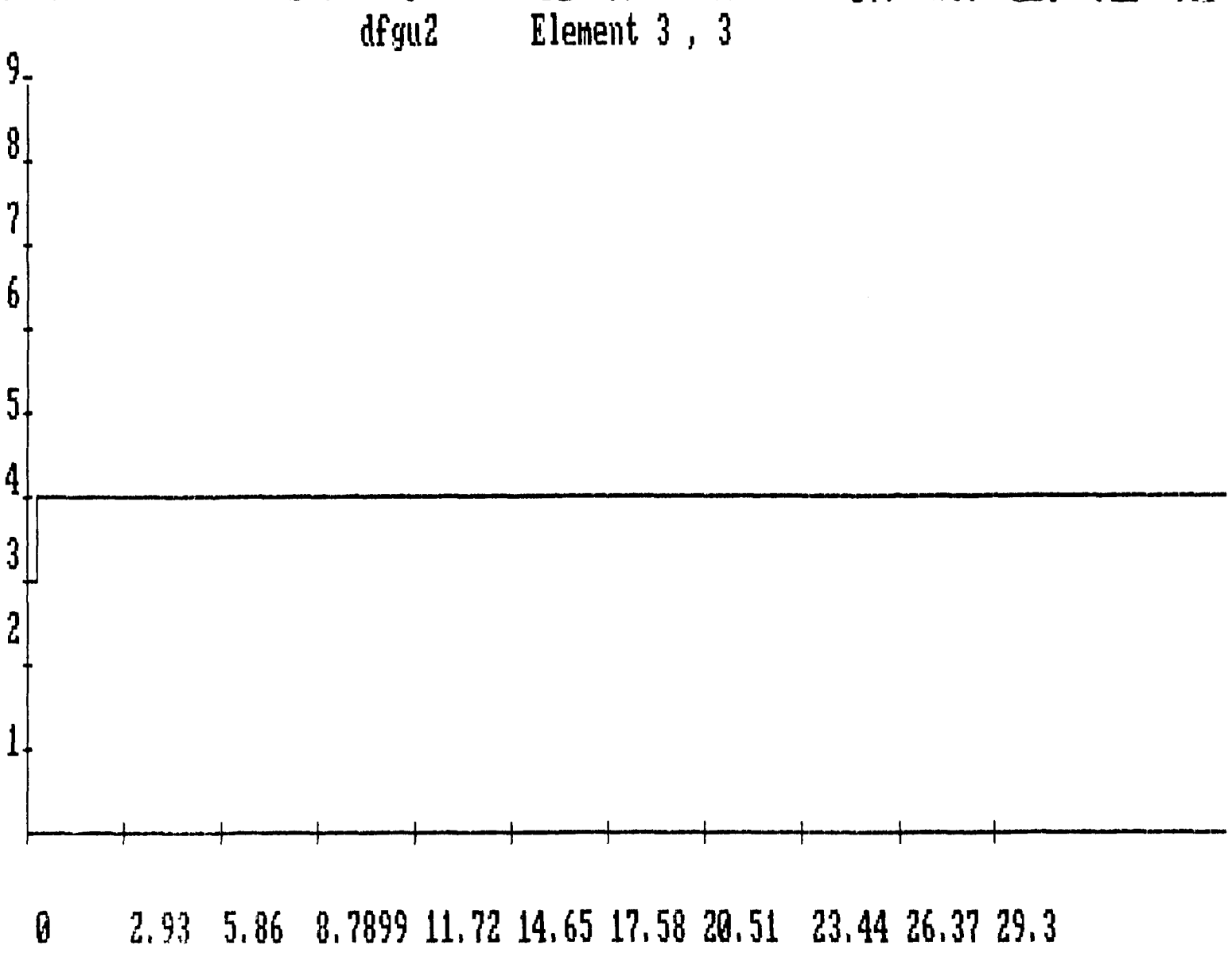

dfgu2 Element 3 , 3
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 13.

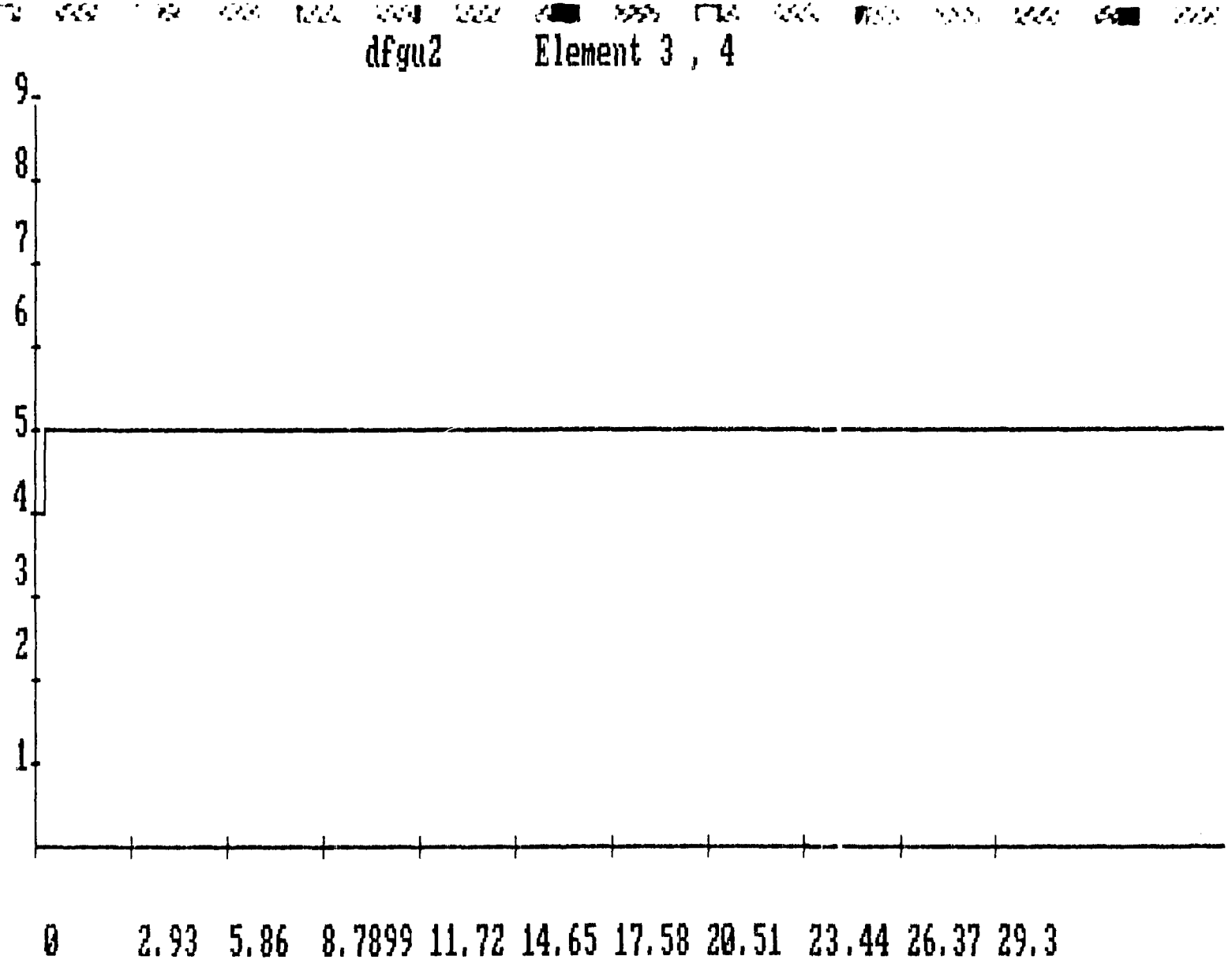

dfgu2 Element 3 , 4
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 14.

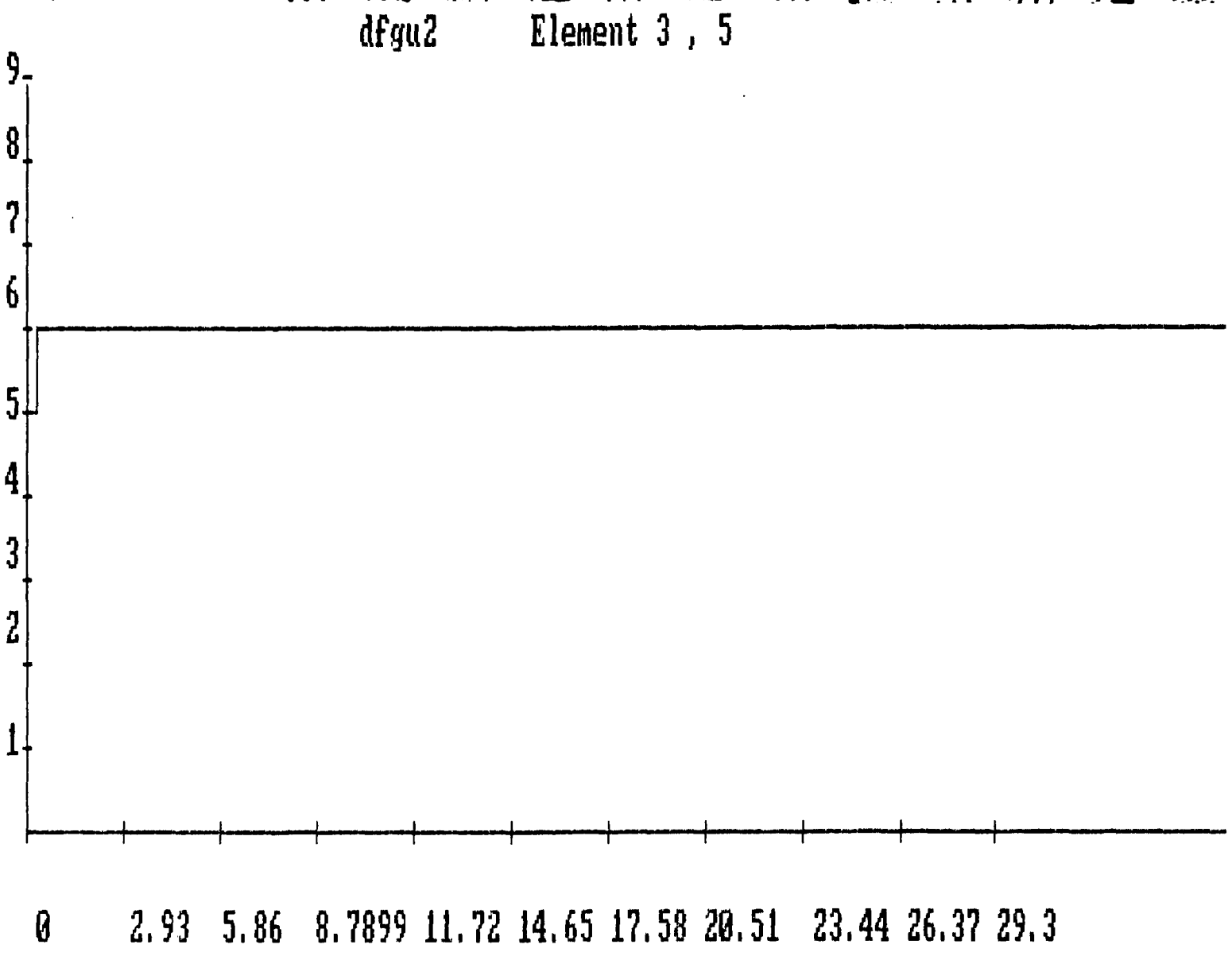

dfgu2 Element 3 , 5
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 15.

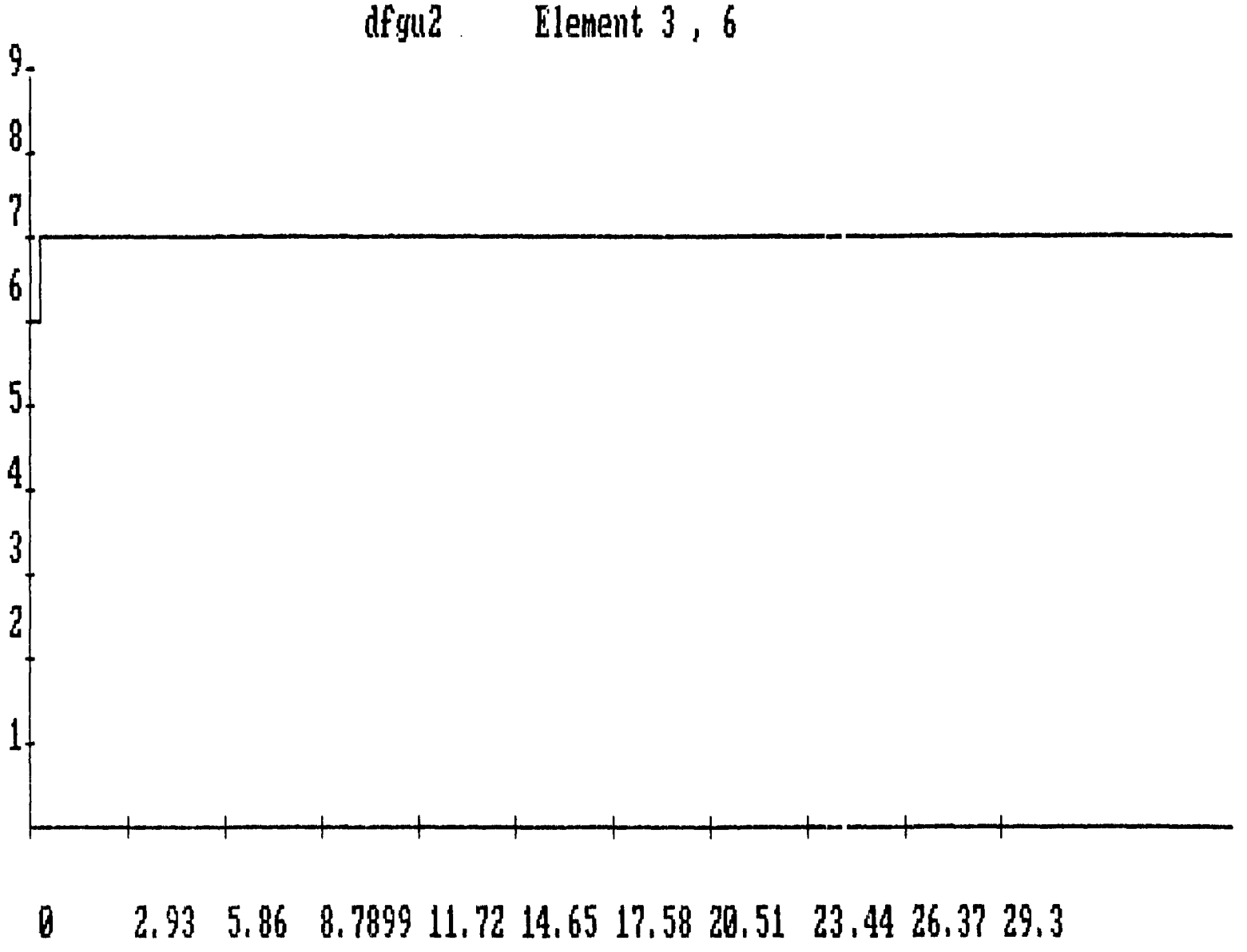

dfgu2 Element 3 , 6
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

Figure 16.

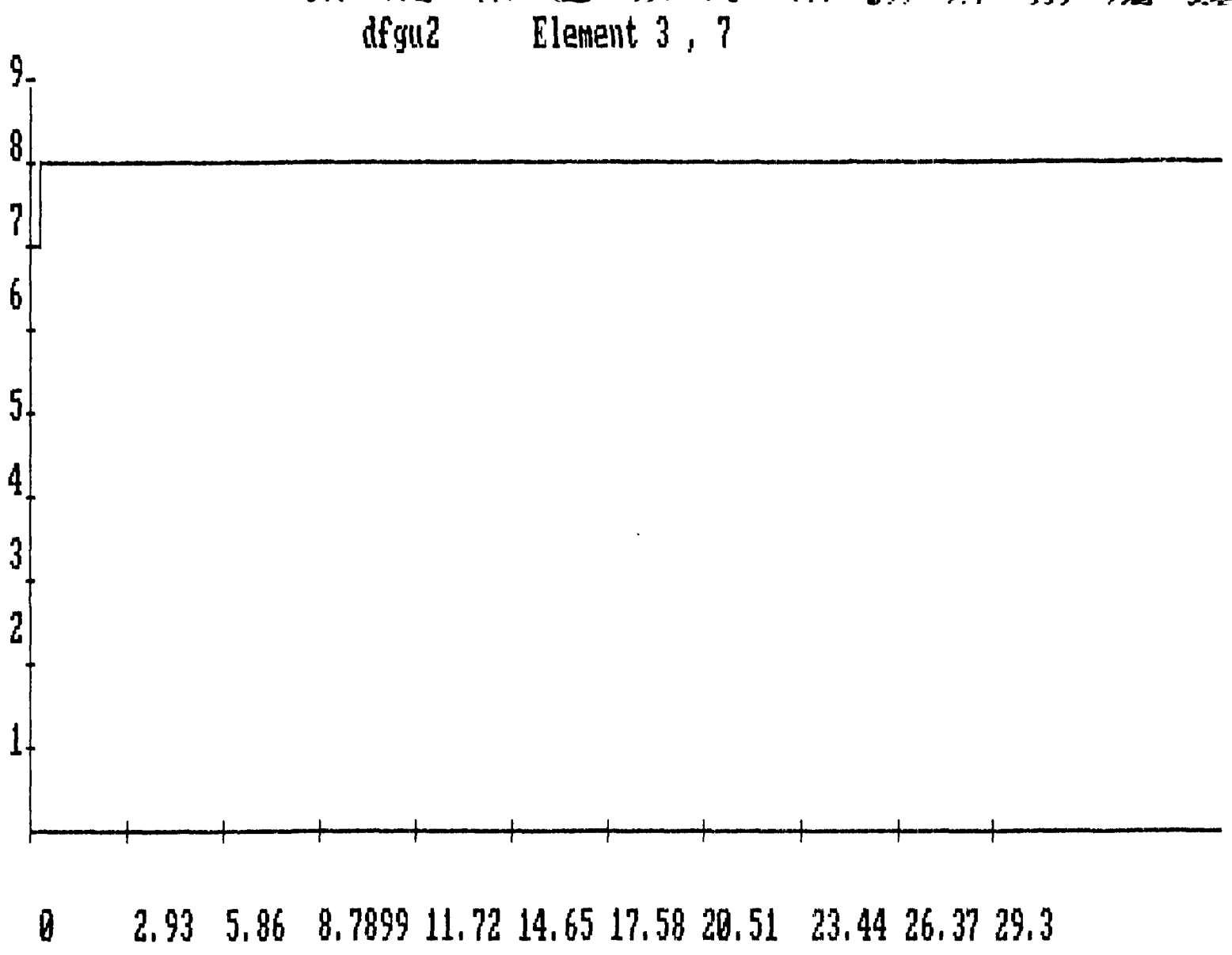

dfgu2 Element 3 , 7
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 17.

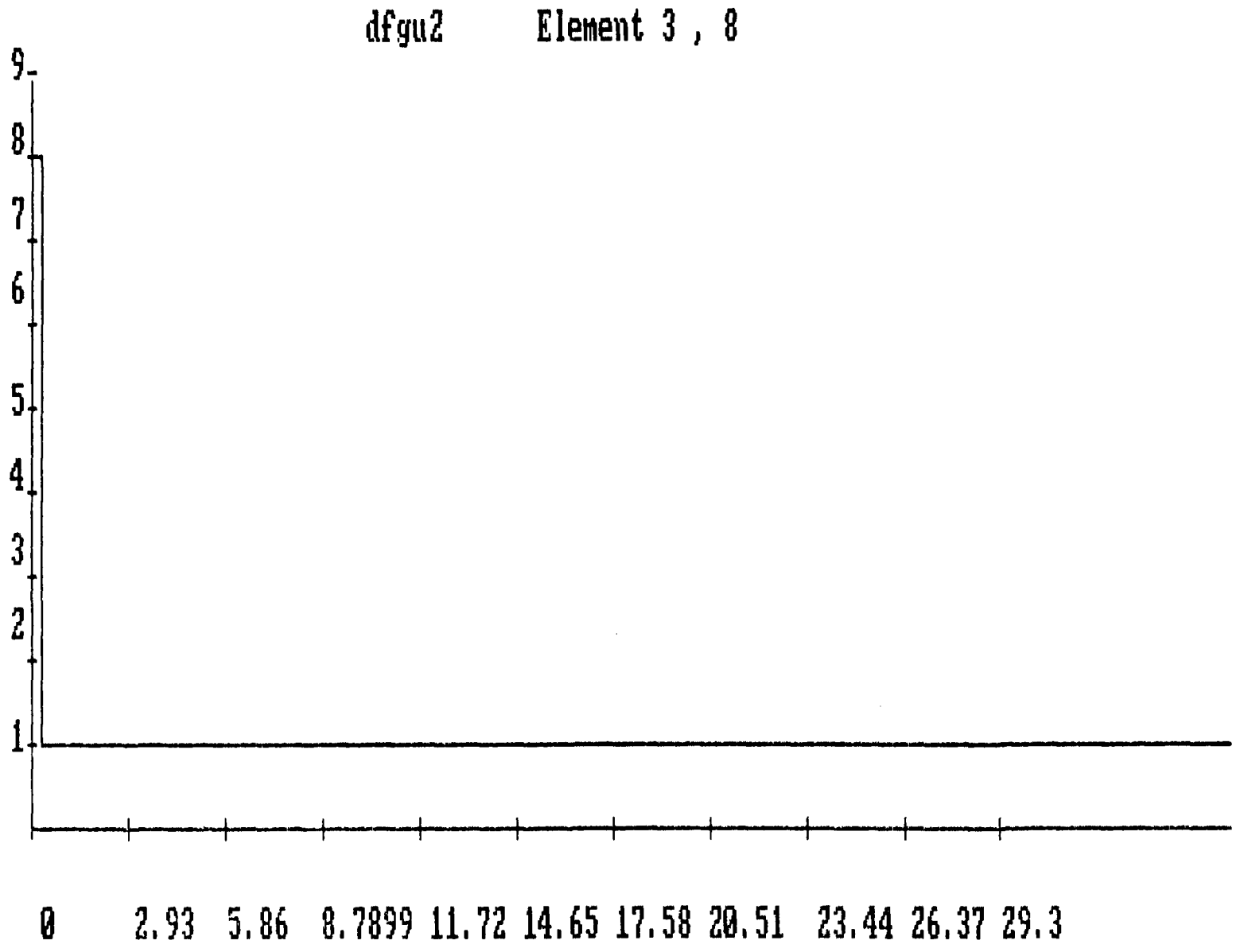

dfgu2 Element 3 , 8
9
8
7
6
5
4
3
2
1
0 2.93 5.86 8.7899 11.72 14.65 17.58 20.51 23.44 26.37 29.3

figure 18.

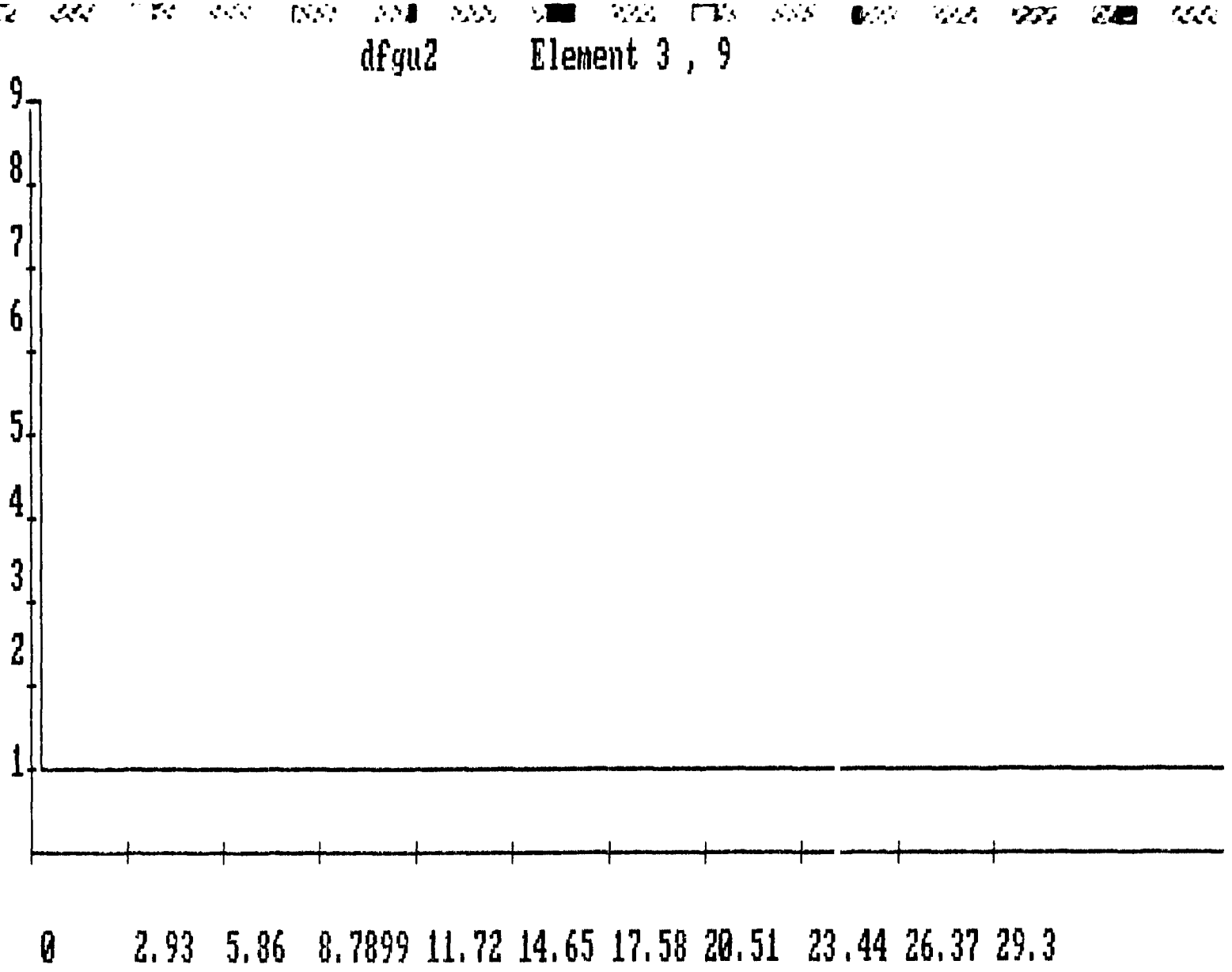

dfgu2 Element 3 , 9
9
8
7
6
5
4
3
2
1
0 2.93 5.86 8.7899 11.72 14.65 17.58 20.51 23.44 26.37 29.3

figure 19.

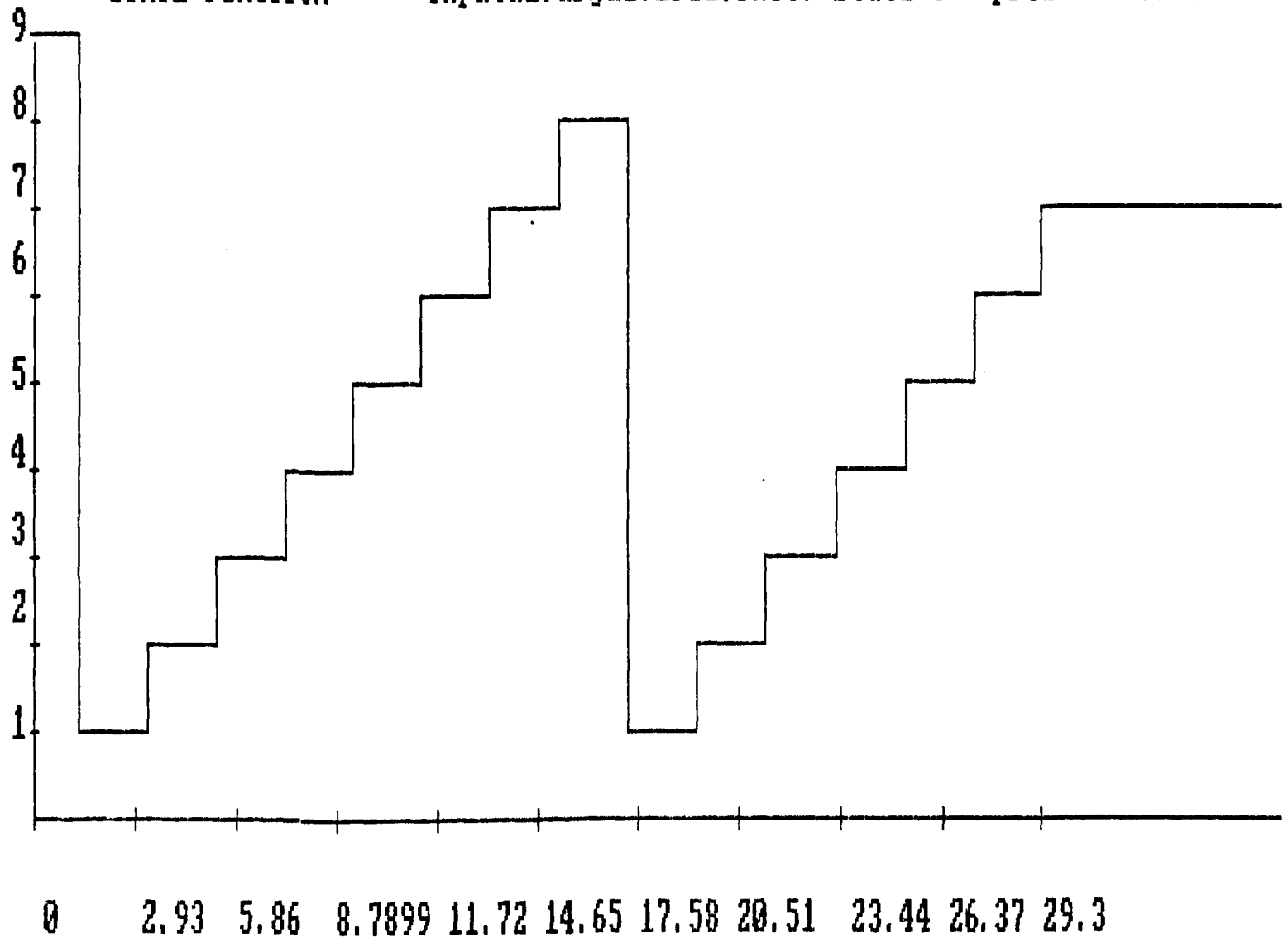

STATE FUNCTION inputu2.dfgu2.LFU2.Init. state 9 proc. time: 0
9
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

figure 20.

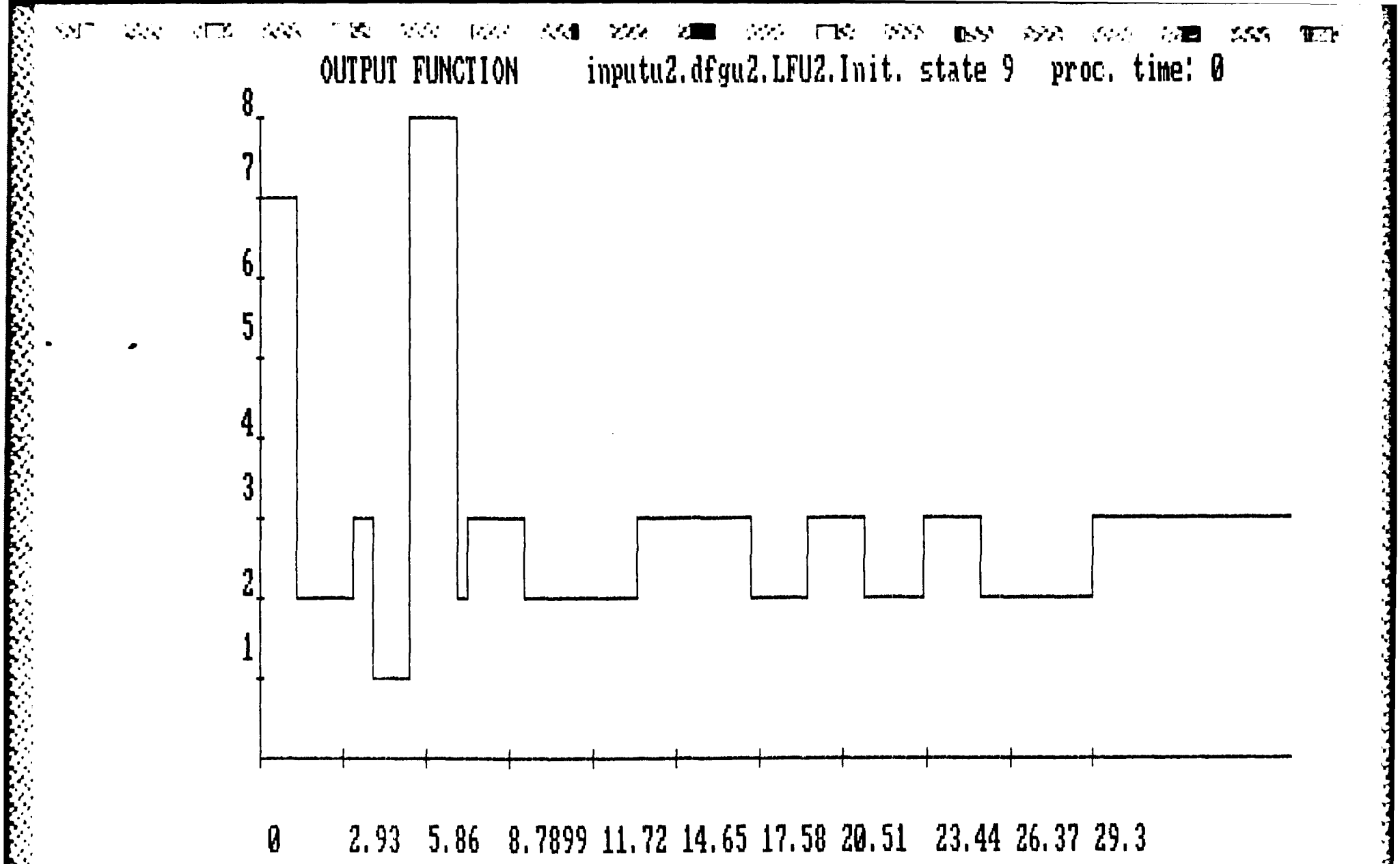
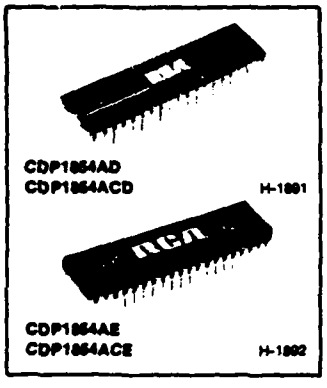

OUTPUT FUNCTION inputu2.dfgu2.LFU2.Init. state 9 proc. time: 0
8
7
6
5
4
3
2
1
0
2.93
5.86
8.7899
11.72
14.65
17.58
20.51
23.44
26.37
29.3

Figure 21.

CDP1854, CDP1854C



Programmable Universal Asynchronous Receiver/Transmitter (UART)

Features:

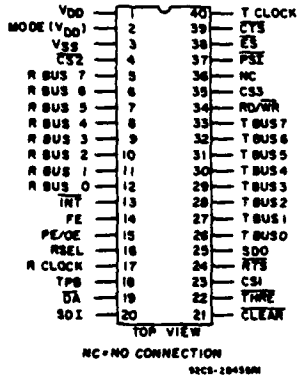
- *Two operating modes:*
  Mode 0–functionally compatible with industry types such as the TR1602A
  Mode 1–interfaces directly with CDP1800-series microprocessors without additional components
- *Full- or half-duplex operation*
- *Parity, framing, and overrun error detection*
- *Baud rate–DC to 200 K bits/sec @ V_{DD}=5 V*
  DC to 400 K bits/sec @ V_{DD}=10 V
- *Fully programmable with externally selectable word length (5-8 bits), parity inhibit, even/odd parity, and 1, 1½, or 2 stop bits*
- *False start bit detection*

The RCA CDP1854A and CDP1854AC are silicon-gate CMOS Universal Asynchronous Receiver/Transmitter (UART) circuits. They are designed to provide the necessary formatting and control for interfacing between serial and parallel data. For example, these UARTs can be used to interface between a peripheral or terminal with serial I/O ports and the 8-bit CDP1800-series microprocessor parallel data bus system. The CDP1854A is capable of full duplex operation, i.e., simultaneous conversion of serial input data to parallel output data and parallel input data to serial output data.

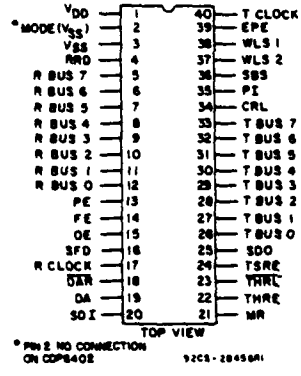
The CDP1854A UART can be programmed to operate in one of two modes by using the mode control input. When the mode input is high (MODE=1), the CDP1854A is directly compatible with the CDP1800-series microprocessor system without additional interface circuitry. When the mode input is low (MODE=0), the device is functionally compatible with industry standard UART's such as the TR1602A. It is also pin compatible with these types, except that pin 2 is used for the mode control input instead of a V_{GG}=−12 V supply connection.

The CDP1854A and the CDP1854AC are functionally identical. The CDP1854A has a recommended operating-voltage range of 4-10.5 volts, and the CDP1854AC has a recommended operating-voltage range of 4-6.5 volts.

The CDP1854A and CDP1854AC are supplied in hermetic 40-lead dual-in-line ceramic packages (D suffix) and in 40-lead dual-in-line plastic packages (E suffix). The *CDP1854AC is also available in chip form* (H suffix).



Mode 1
Terminal Assignment



Mode 0
Terminal Assignment

MAXIMUM RATINGS, *Absolute-Maximum Values:*

DC SUPPLY-VOLTAGE RANGE, (V_{DD})
(Voltage referenced to V_{SS} Terminal)
- CDP1854A ..... −0.5 to +11 V
- CDP1854AC ..... −0.5 to +7 V

INPUT VOLTAGE RANGE, ALL INPUTS ..... −0.5 to V_{DD} +0.5 V
DC INPUT CURRENT, ANY ONE INPUT ..... ±10 mA
POWER DISSIPATION PER PACKAGE (P_D):
- For T_A=−40 to +60°C (PACKAGE TYPE E) ..... 500 mW
- For T_A=+60 to +85°C (PACKAGE TYPE E) ..... Derate Linearly at 12 mW/°C to 200 mW
- For T_A=−55 to 100°C (PACKAGE TYPE D) ..... 500 mW
- For T_A=+100 to +125°C (PACKAGE TYPE D) ..... Derate Linearly at 12 mW/°C to 200 mW

DEVICE DISSIPATION PER OUTPUT TRANSISTOR
- FOR T_A=FULL PACKAGE-TEMPERATURE RANGE (All Package Types) ..... 100 mW

OPERATING-TEMPERATURE RANGE (T_A):
- PACKAGE TYPE D ..... −55 to +125°C
- PACKAGE TYPE E ..... −40 to +85°C

STORAGE TEMPERATURE RANGE (T_{stg}) ..... −65 to +150°C
LEAD TEMPERATURE (DURING SOLDERING):
- At distance 1/16 ± 1/32 in. (1.59 ± 0.79 mm) from case for 10 s max. ..... +265°C

Mode Input High (Mode = 1)

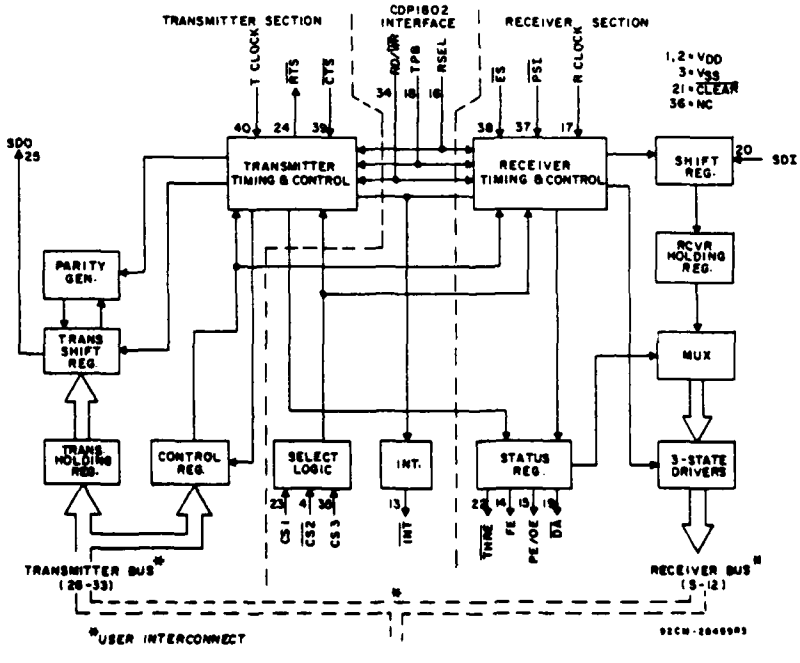


Fig. 1 - Mode 1 block diagram (CDP1800-series microprocessor compatible).

CDP1854, CDP1854C

STATIC ELECTRICAL CHARACTERISTICS at T_A = -40 to +85°C, unless otherwise noted.

CHARACTERISTIC	CONDITIONS			LIMITS						UNITS
	V_O (V)	V_{IN} (V)	V_{DD} (V)	CDP1854A Min.	CDP1854A Typ.*	CDP1854A Max.	CDP1854AC Min.	CDP1854AC Typ.*	CDP1854AC Max.	
Quiescent Device Current, I_{DD}	—	0, 5	5	—	0.01	50	—	0.02	200	μA
	—	0, 10	10	—	1	200	—	—	—	
Output Low Drive (Sink) Current, I_{OL}	0.4	0, 5	5	0.55	1.1	—	0.55	1.1	—	mA
	0.5	0, 10	10	1.3	2.6	—	—	—	—	
Output High Drive (Source) Current, I_{OH} (Except pins 24 and 25)	4.6	0, 5	5	−0.55	−1.1	—	−0.55	−1.1	—	mA
	9.5	0, 10	10	−1.3	−2.6	—	—	—	—	
Output High Drive (Source) Current, I_{OH} Pins 24 and 25	4.6	0, 5	5	−1.6	−3.5	—	−1.6	−3.5	—	mA
	9.5	0, 10	10	−2.8	−6.0	—	—	—	—	
Output Voltage Low-Level, V_{OL}*	—	0, 5	5	—	0	0.1	—	0	0.1	V
	—	0, 10	10	—	0	0.1	—	—	—	
Output Voltage High-Level, V_{OH}	—	0, 5	5	4.9	5	—	4.9	5	—	V
	—	0, 10	10	9.9	10	—	—	—	—	
Input Low Voltage, V_{IL}	0.5,4.5	—	5	—	—	1.5	—	—	1.5	V
	0.5,9.5	—	10	—	—	3	—	—	—	
Input High Voltage, V_{IH}	0.5,4.5	—	5	3.5	—	—	3.5	—	—	V
	0.5,9.5	—	10	7	—	—	—	—	—	
Input Current, I_{IN}	—	0, 5	5	—	—	±1	—	—	±1	μA
	—	0, 10	10	—	—	±2	—	—	—	
3-State Output Leakage Current, I_{OUT}	0, 5	0, 5	5	—	—	±1	—	—	±1	μA
	0, 10	0, 10	10	—	—	±10	—	—	—	
Operating Current, I_{DD1}#	—	0, 5	5	—	1.5	3.0	—	1.5	3.0	mA
	—	0, 10	10	—	6.0	12	—	—	—	
Input Capacitance, C_{IN}	—	—	—	—	5	7.5	—	5	7.5	pF
Output Capacitance, C_{OUT}	—	—	—	—	10	15	—	10	15	

*Typical values are for T_A=25°C. *I_{OL}=I_{OH}=1 μA.

#Operating current is measured at 200 kHz for V_{DD}=5 V and 400 kHz for V_{DD}=10 V in a CDP1800-series microprocessor system, with open outputs.

RECOMMENDED OPERATING CONDITIONS at T_A=Full Package Temperature Range

For maximum reliability, operating conditions should be selected so that operation is always within the following ranges:

CHARACTERISTIC	CONDITIONS	LIMITS				UNITS
	V_{DD} V	CDP1854A Min.	CDP1854A Max.	CDP1854AC Min.	CDP1854AC Max.	
DC Operating-Voltage Range	—	4	10.5	4	6.5	V
Input Voltage Range	—	V_{SS}	V_{DD}	V_{SS}	V_{DD}	V
Baud Rate (Receive or Transmit)	5	—	200	—	200	K bits
	10	—	400	—	—	/sec

Functional Definitions for CDP1854A Terminals
Mode 1
CDP1800-Series Microprocessor Compatible
SIGNAL: FUNCTION

V_{DD}:
Positive supply voltage

MODE SELECT (MODE):
A high-level voltage at this input selects CDP1800-series microprocessor Mode operation.

V_{SS}:
Ground

CHIP SELECT 2 ($\overline{CS2}$):
A low-level voltage at this input together with CS1 and CS3 selects the CDP1854A UART.

RECEIVER BUS (R BUS 7 - R BUS 0):
Receiver parallel data outputs (may be externally connected to corresponding transmitter bus terminals).

$\overline{INTERRUPT}$ ($\overline{INT}$):
A low-level voltage at this output indicates the presence of one or more of the interrupt conditions listed in Table I.

FRAMING ERROR (FE):
A high-level voltage at this output indicates that the received character has no valid stop bit, i.e., the bit following the parity bit (if programmed) is not a high-level voltage. This output is updated each time a character is transferred to the Receiver Holding Register.

PARITY ERROR or OVERRUN ERROR (PE/OE):
A high-level voltage at this output indicates that either the PE or OE bit in the Status Register has been set (see Status Register Bit Assignment, Table II.

REGISTER SELECT (RSEL):
This input is used to choose either the Control/Status Registers (high input) or the transmitter/receiver data registers (low input) according to the truth table in Table III.

RECEIVER CLOCK (RCLOCK):
Clock input with a frequency 16 times the desired receiver shift rate.

TPB:
A positive input pulse used as a data load or reset strobe.

$\overline{DATA\ AVAILABLE}$ ($\overline{DA}$):
A low-level voltage at this output indicates that an entire character has been received and transferred to the Receiver Holding Register.

SERIAL DATA IN (SDI):
Serial data received on this input line enters the Receiver Shift Register at a point determined by the character length. A high-level input voltage must be present when data is not being received.

$\overline{CLEAR}$ ($\overline{CLEAR}$):
A low-level voltage at this input resets the Interrupt Flip-Flop, Receiver Holding Register, Control Register, and Status Register, and sets SERIAL DATA OUT (SDO) high.

$\overline{TRANSMITTER\ HOLDING\ REGISTER\ EMPTY}$ ($\overline{THRE}$):
A low-level voltage at this output indicates that the Transmitter Holding Register has transferred its contents to the Transmitter Shift Register and may be reloaded with a new character.

CHIP SELECT 1 (CS1):
A high-level voltage at this input together with $\overline{CS2}$ and CS3 selects the UART.

$\overline{REQUEST\ TO\ SEND}$ ($\overline{RTS}$):
This output signal tells the peripheral to get ready to receive data. $\overline{CLEAR\ TO\ SEND}$ ($\overline{CTS}$) is the response from the peripheral. $\overline{RTS}$ is set to a low-level voltage when data is latched in the Transmitter Holding Register or TR is set high, and is reset high when both the Transmitter Holding Register and Transmitter Shift Register are empty and TR is low.

SERIAL DATA OUTPUT (SDO):
The contents of the Transmitter Shift Register (start bit, data bits, parity bit, and stop bit(s) are serially shifted out on this output. When no character is being transmitted, a high level is maintained. Start of transmission is defined as the transition of the start bit from a high-level to a low-level output voltage.

TRANSMITTER BUS (T BUS 0 - T BUS 7):
Transmitter parallel data input. These may be externally connected to corresponding Receiver bus terminals.

RD/$\overline{WR}$:
A low-level voltage at this input gates data from the transmitter bus to the Transmitter Holding Register or the Control Register as chosen by register select. A high-level voltage gates data from the Receiver Holding Register or the Status Register, as chosen by register select, to the receiver bus.

CHIP SELECT 3 (CS3):
With high-level voltage at this input together with CS1 and $\overline{CS2}$ selects the UART.

$\overline{PERIPHERAL\ STATUS\ INTERRUPT}$ ($\overline{PSI}$):
A high-to-low transition on this input line sets a bit in the Status Register and causes an INTERRUPT ($\overline{INT}$=low).

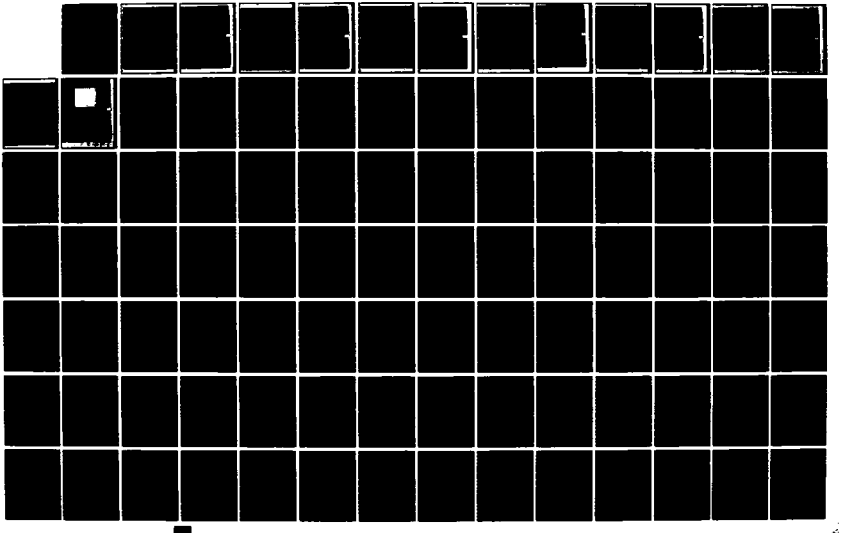
$\overline{EXTERNAL\ STATUS}$ ($\overline{ES}$):
A low-level voltage at this input sets a bit in the Status Register.

$\overline{CLEAR\ TO\ SEND}$ ($\overline{CTS}$):
When this input from peripheral is high, transfer of a character to the Transmitter Shift Register and shifting of serial data out is inhibited.

TRANSMITTER CLOCK (TCLOCK):
Clock input with a frequency 16 times the desired transmitter shift rate.

AD-A174 525 ASYNCHRONOUS DISCRETE CONTROL OF CONTINUOUS PROCESSES (U) NORTHEASTERN UNIV BOSTON MA M E KALISKI 24 FEB 86 AFOSR-TR-86-2052 F49620-82-C-0008 2/3

UNCLASSIFIED F/G 9/4 NL



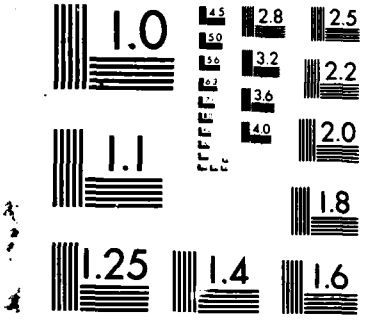


Table I — Interrupt Set and Reset Conditions

SET* (INT = LOW)	RESET (INT = HIGH)	
CAUSE	CONDITION	TIME
DA (Receipt of data)	Read of data	TPB leading edge
THRE* (Ability to reload)	Read of status or write of character	TPB leading edge
THRE · TSRE (Transmitter done)	Read of status or write of character	TPB leading edge
$\overline{PSI}$ (Negative edge)	Read of status	TPB trailing edge
$\overline{CTS}$ (Positive edge when $\overline{THRE \cdot TSRE}$)	Read of status	TPB leading edge

*Interrupts will occur only after the IE bit in the Control Register (see Table IV) has been set.
**THRE will cause an interrupt only after the TR bit in the Control Register (see Table IV) has been set.

Table II — Status Register Bit Assignment

Bit	7	6	5	4	3	2	1	0
Signal	THRE	TSRE	PSI	ES	FE	PE	OE	DA
Also Available at Terminal	22*	—	—	—	14	15	15	19*

*Polarity reversed at output terminal.

Bit Signal: Function

0—DATA AVAILABLE (DA):
When set high, this bit indicates that an entire character has been received and transferred to the Receiver Holding Register. This signal is also available at Term. 19 but with its polarity reversed.

1—OVERRUN ERROR (OE):
When set high, this bit indicates that the Data Available bit was not reset before the next character was transferred to the Receiver Holding Register. This signal OR'ed with PE is output at Term. 15.

2—PARITY ERROR (PE):
When set high, this bit indicates that the received parity bit does not compare to that programmed by the EVEN PARITY ENABLE (EPE) control. This bit is updated each time a character is transferred to the Receiver Holding Register. This signal OR'ed with OE is output at Term. 15.

3—FRAMING ERROR (FE):
When set high, this bit indicates that the received character has no valid stop bit, i.e., the bit following the parity bit (if programmed) is not a high-level voltage. This bit is updated each time a character is transferred to the Receiver Holding Register. This signal is also available at Term. 14.

4—EXTERNAL STATUS (ES):
This bit is set high by a low-level input at Term. 38 ($\overline{ES}$).

5—PERIPHERAL STATUS INTERRUPT (PSI):
This bit is set high by a high-to-low voltage transition of Term. 37 ($\overline{PSI}$). The INTERRUPT output (Term. 13) is also asserted (INT=low) when this bit is set.

6—TRANSMITTER SHIFT REGISTER EMPTY (TSRE):
When set high, this bit indicates that the Transmitter Shift Register has completed serial transmission of a full character including stop bit(s). It remains set until the start of transmission of the next character.

7—TRANSMITTER HOLDING REGISTER EMPTY (THRE):
When set high, this bit indicates that the Transmitter Holding Register has transferred its contents to the Transmitter Shift Register and may be reloaded with a new character. Setting this bit also sets the $\overline{THRE}$ output (Term. 22) low and causes an INTERRUPT ($\overline{INT}$=low), if TR is high.

Description of Mode 1 Operation CDP1800-Series Microprocessor Compatible (Mode Input=V_{DD})

1. Initialization and Controls

In the CDP1800-series microprocessor compatible mode, the CDP1854A is configured to receive commands and send status via the microprocessor data bus. The register connected to the transmitter bus or the receiver bus is determined by the RD/$\overline{WR}$ and RSEL inputs as follows:

Table III — Register Selection Summary

RSEL	RD/$\overline{WR}$	Function
Low	Low	Load Transmitter Holding Register from Transmitter Bus
Low	High	Read Receiver Holding Register from Receiver Bus
High	Low	Load Control Register from Transmitter Bus
High	High	Read Status Register from Receiver Bus

In this mode the CDP1854A is compatible with a bidirectional bus system. The receiver and transmitter buses are connected to the bus. CDP1800-series microprocessor I/O control output signals can be connected directly to the CDP1854A inputs as shown in Fig. 2. The CLEAR input is pulsed, resetting the Control, Status, and Receiver Holding Registers and setting SERIAL DATA OUT (SDO) high. The Control Register is loaded from the Transmitter Bus in order to determine the operating configuration for the UART. Data is transferred from the Transmitter Bus inputs to the Control Register during TPB when the UART is selected CS1 · $\overline{CS2}$ · CS3=1) and the Control Register is designated (RSEL=H, RD/$\overline{WR}$=L). The CDP1854A also has a Status Register which can be read onto the Receiver Bus (R BUS 0 - R BUS 7) in order to determine the status of the UART. Some of these status bits are also available at separate terminals as indicated in Table II.

2. Transmitter Operation

Before beginning to transmit, the TRANSMIT REQUEST (TR) bit in the Control Register (see bit assignment, Table IV) is set. Loading the Control Register with TR=1 (bit 7=high) inhibits changing the other control bits. Therefore two loads are required: one to format the UART, the second to set TR. When TR has been set, a TRANSMITTER HOLDING REGISTER EMPTY ($\overline{THRE}$) interrupt will occur, signalling the microprocessor that the Transmitter Holding Register is empty and may be loaded. Setting TR also causes assertion of a low-level on the REQUEST TO SEND ($\overline{RTS}$) output to the peripheral. It is not necessary to set TR for proper operation for the UART. If desired, it can be used to enable $\overline{THRE}$ interrupts and to generate the $\overline{RTS}$ signal. The Transmitter Holding Register is loaded from the bus by TPB during execution of an output instruction. The CDP1854A is selected by CS1 · $\overline{CS2}$ · CS3=1, and the Holding Register is selected by RSEL=L and RD/$\overline{WR}$=L. When the CLEAR TO SEND ($\overline{CTS}$) input, which can be connected to a peripheral device output, goes low, the Transmitter Shift Register will be loaded from the Transmitter Holding Register and data transmission will begin. If $\overline{CTS}$ is always low, the Transmitter Shift Register will be loaded on the first high-to-low edge of the clock which occurs at least 1/2 clock period after the trailing edge of TPB and transmission of a start bit will occur 1/2 clock period later (see Fig. 3). Parity (if programmed) and stop bit(s) will be transmitted following the last data bit. If the word length selected is less than 8 bits, the most significant unused bits in the transmitter shift register will not be transmitted.

One transmitter clock period after the Transmitter Shift Register is loaded from the Transmitter Holding Register, the $\overline{THRE}$ signal will go low and an interrupt will occur ($\overline{INT}$ goes low). The next character to be transmitted can then be loaded into the Transmitter Holding Register for transmission with its start bit immediately following the last stop bit of the previous character. This cycle can be repeated until the last character is transmitted, at which time a final THRE · TSRE interrupt will occur. This interrupt signals the microprocessor that TR can be turned off. This is done by reloading the original control byte in the Control Register with the TR bit = 0, thus terminating the $\overline{REQUEST\ TO\ SEND}$ ($\overline{RTS}$) signal.

SERIAL DATA OUT (SDO) can be held low by setting the BREAK bit in the Control Register (see Table IV). SDO is held low until the BREAK bit is reset.

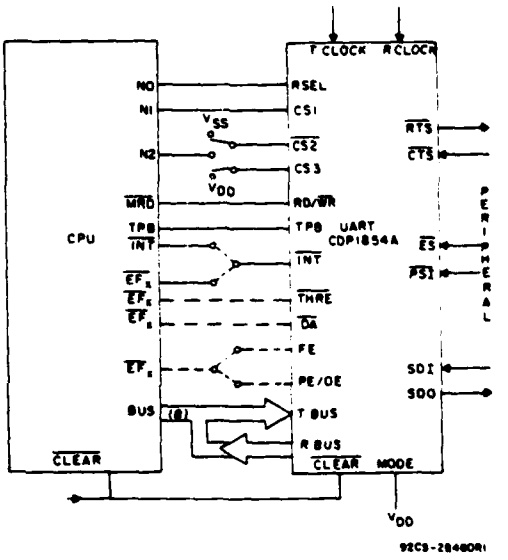


Fig. 2 - Recommended CDP1800-series connection, Mode 1 (non-interrupt driven system).

3. Receiver Operation

The receive operation begins when a start bit is detected at the SERIAL DATA IN (SDI) input. After detection of the first high-to-low transition on the SDI line, a valid start bit is verified by checking for a low-level input 7-1/2 receiver clock periods later. When a valid start bit has been verified, the following data bits, parity bit (if programmed) and stop bit(s) are shifted into the Receiver Shift Register by clock pulse 7-1/2 in each bit time. The parity bit (if programmed) is checked and receipt of a valid stop bit is verified. On count 7-1/2 of the first stop bit, the received data is loaded into the Receiver Holding Register. If the word length is less than 8 bits, zeros (low output level) are loaded into the unused most significant bits. If DATA AVAILABLE (DA) has not been reset by the time the Receiver Holding Register is loaded, the OVERRUN ERROR (OE) status bit is set. One half clock period later, the PARITY ERROR (PE) and FRAMING ERROR (FE) status bits become valid for the character in the Receiver Holding Register. At this time, the Data Available status bit is also set and the DATA AVAILABLE (DA) and INTERRUPT (INT) outputs go low, signalling the microprocessor that a received character is

ready. The microprocessor responds by executing an input instruction. The UART's 3-state bus drivers are enabled when the UART is selected (CS1 · $\overline{CS2}$ · CS3=1) and RD/$\overline{WR}$=high. Status can be read when RSEL=high. Data is read when RSEL=low. When reading data, TPB latches data in the microprocessor and resets DATA AVAILABLE ($\overline{DA}$) in the UART. The preceding sequence is repeated for each serial character which is received from the peripheral.

4. Peripheral Interface

In addition to serial data in and out, four signals are provided for communication with a peripheral. The $\overline{REQUEST\ TO\ SEND}$ ($\overline{RTS}$) output signal alerts the peripheral to get ready to receive data. The $\overline{CLEAR\ TO\ SEND}$ ($\overline{CTS}$) input signal is the response, signalling that the peripheral is ready. The EXTERNAL $\overline{STATUS}$ (ES) input latches a peripheral status level, and the $\overline{PERIPHERAL\ STATUS\ INTERRUPT}$ ($\overline{PSI}$) input senses a status edge (high-to-low) and also generates an interrupt. For example, the modem $\overline{DATA\ CARRIER\ DETECT}$ line could be connected to the $\overline{PSI}$ input on the UART in order to signal the microprocessor that transmission failed because of loss of the carrier on the communications line. The PSI and ES bits are stored in the Status Register (see Table II).

Table IV — Control Register Bit Assignment

Bit	7	6	5	4	3	2	1	0
Signal	TR	BREAK	IE	WLS2	WLS1	SBS	EPE	PI

Bit Signal: Function

0—PARITY INHIBIT (PI):
When set high parity generation and verification are inhibited and the PE Status bit is held low. If parity is inhibited the stop bit(s) will immediately follow the last data bit on transmission, and EPE is ignored.

1—EVEN PARITY ENABLE (EPE):
When set high, even parity is generated by the transmitter and checked by the receiver. When low, odd parity is selected.

2—STOP BIT SELECT (SBS):
See table below.

3—WORD LENGTH SELECT 1 (WLS1):
See table below.

4—WORD LENGTH SELECT 2 (WLS2):
See table below.

5—INTERRUPT ENABLE (IE):
When set high $\overline{THRE}$, DA, THRE · TSRE, $\overline{CTS}$, and PSI interrupts are enabled (see Interrupt Conditions, Table I).

6—TRANSMIT BREAK (BREAK):
Holds SDO low when set. Once the break bit in the control register has been set high, SDO will stay low until the break bit is reset low and one of the following occurs: $\overline{CLEAR}$ goes low; $\overline{CTS}$ goes high; or a word is transmitted. (The transmitted word will not be valid since there can be no start bit if SDO is already low. SDO can be set high without intermediate transitions by transmitting a word consisting of all zeros).

7—TRANSMIT REQUEST (TR):
When set high, $\overline{RTS}$ is set low and data transfer through the transmitter is initiated by the initial $\overline{THRE}$ interrupt. (When loading the Control Register from the bus, this (TR) bit inhibits changing of other control flip-flops).

Bit 4	Bit 3	Bit 2	Function
WLS2	WLS1	SBS	
0	0	0	5 data bits, 1 stop bit
0	0	1	5 data bits, 1.5 stop bits
0	1	0	6 data bits, 1 stop bit
0	1	1	6 data bits, 2 stop bits
1	0	0	7 data bits, 1 stop bit
1	0	1	7 data bits, 2 stop bits
1	1	0	8 data bits, 1 stop bit
1	1	1	8 data bits, 2 stop bits

DYNAMIC ELE
C_L=100 pF, see

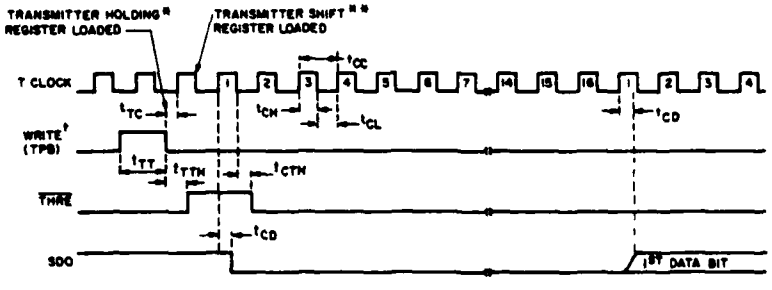
Transmitter Tim
Minimum Clock
Minimum Pulse
Clock Low Le
Clock High Le
TPB
Minimum Setup
TPB to Clock
Propagation De
Clock to Data
TPB to $\overline{THRE}$
Clock to $\overline{THR}$

†Typical values ar
*Maximum limits

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = -40 to +85°C, V_{DD} ±5%, t_r,t_f=20 ns, V_{IH}=0.7 V_{DD}, V_{IL}=0.3 V_{DD}, C_L=100 pF, see Fig. 3.

CHARACTERISTIC		V_{DD} (V)	LIMITS				UNITS
			CDP1854A		CDP1854AC		
			Typ.†	Max.*	Typ.†	Max.*	
Transmitter Timing — Mode 1							
Minimum Clock Period	t_{CC}	5	250	310	250	310	ns
		10	125	155	—	—	
Minimum Pulse Width:		5	100	125	100	125	ns
Clock Low Level	t_{CL}	10	75	100	—	—	
Clock High Level	t_{CH}	5	100	125	100	125	ns
		10	75	100	—	—	
TPB	t_{TT}	5	100	150	100	150	ns
		10	50	75	—	—	
Minimum Setup Time:		5	175	225	175	225	ns
TPB to Clock	t_{TC}	10	90	150	—	—	
Propagation Delay Time:		5	300	450	300	450	ns
Clock to Data Start Bit	t_{CD}	10	150	225	—	—	
TPB to $\overline{THRE}$	t_{TTH}	5	200	300	200	300	ns
		10	100	150	—	—	
Clock to $\overline{THRE}$	t_{CTH}	5	200	300	200	300	ns
		10	100	150	—	—	

†Typical values are for T_A=25°C and nominal voltages.
*Maximum limits of minimum characteristics are the values above which all devices function.



* THE HOLDING REGISTER IS LOADED ON THE TRAILING EDGE OF TPB.
* * THE TRANSMITTER SHIFT REGISTER IS LOADED ON THE FIRST HIGH-TO-LOW TRANSITION OF THE CLOCK WHICH OCCURS AT LEAST 1/2 CLOCK PERIOD + t_{TC} AFTER THE TRAILING EDGE OF TPB, AND TRANSMISSION OF A START BIT OCCURS 1/2 CLOCK PERIOD + t_{CD} LATER.
† WRITE IS THE OVERLAP OF TPB, CS1, AND CS3 = 1 AND $\overline{CS2}$, RD / $\overline{WR}$ = 0.

92CM-31678

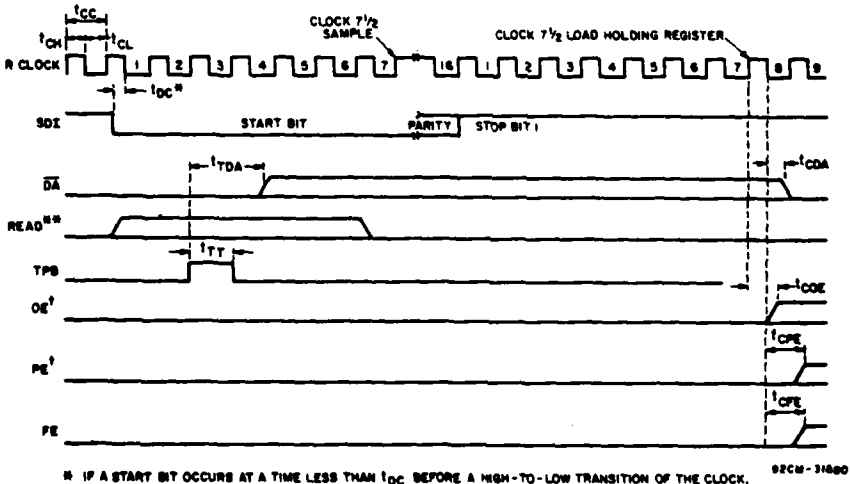
Fig. 3 - Transmitter timing diagram - Mode 1.

CDP1854, CDP1854C

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = -40 to +85°C, V_{DD} ±5%, t_r, t_f=20 ns, V_{IH}=0.7 V_{DD}, V_{IL}=0.3 V_{DD}, C_L=100 pF, see Fig. 4.

CHARACTERISTIC		V_{DD} (V)	LIMITS CDP1854A Typ.†	CDP1854A Max.*	CDP1854AC Typ.†	CDP1854AC Max.*	UNITS
Receiver Timing — Mode 1							
Minimum Clock Period	t_{CC}	5	250	310	250	310	ns
		10	125	155	—	—	
Minimum Pulse Width:							
Clock Low Level	t_{CL}	5	100	125	100	125	ns
		10	75	100	—	—	
Clock High Level	t_{CH}	5	100	125	100	125	ns
		10	75	100	—	—	
TPB	t_{TT}	5	100	150	100	150	ns
		10	50	75	—	—	
Minimum Setup Time:							
Data Start Bit to Clock	t_{DC}	5	100	150	100	150	ns
		10	50	75	—	—	
Propagation Delay Time:							
TPB to $\overline{\text{DATA AVAILABLE}}$	t_{TDA}	5	220	325	220	325	ns
		10	110	175	—	—	
Clock to $\overline{\text{DATA AVAILABLE}}$	t_{CDA}	5	220	325	220	325	ns
		10	110	175	—	—	
Clock to Overrun Error	t_{COE}	5	210	300	210	300	ns
		10	105	150	—	—	
Clock to Parity Error	t_{CPE}	5	240	375	240	375	ns
		10	120	175	—	—	
Clock to Framing Error	t_{CFE}	5	200	300	200	300	ns
		10	100	150	—	—	

†Typical values are for T_A=25°C and nominal voltages.
*Maximum limits of minimum characteristics are the values above which all devices function.



* IF A START BIT OCCURS AT A TIME LESS THAN t_{DC} BEFORE A HIGH-TO-LOW TRANSITION OF THE CLOCK, THE START BIT MAY NOT BE RECOGNIZED UNTIL THE NEXT HIGH-TO-LOW TRANSITION OF THE CLOCK. THE START BIT MAY BE COMPLETELY ASYNCHRONOUS WITH THE CLOCK.

** READ IS THE OVERLAP OF CS1, CS3, RD/$\overline{WR}$=1 AND $\overline{CS2}$=0. IF A PENDING DA HAS NOT BEEN CLEARED BY A READ OF THE RECEIVER HOLDING REGISTER BY THE TIME A NEW WORD IS LOADED INTO THE RECEIVER HOLDING REGISTER, THE OE SIGNAL WILL COME TRUE

† OE AND PE SHARE TERMINAL IS AND ARE ALSO AVAILABLE AS TWO SEPARATE BITS IN THE STATUS REGISTER

Fig. 4 - Mode 1 receiver timing diagram.

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = −40 to +85°C, V_{DD} ±5%, t_r,t_f=20 ns, V_{IH}=0.7 V_{DD}, V_{IL}=0.3 V_{DD}, C_L=100 pF, see Fig. 6.

CHARACTERISTIC		V_{DD} (V)	LIMITS				UNITS
			CDP1854A		CDP1854AC		
			Typ.†	Max.*	Typ.†	Max.*	
CPU Interface — WRITE Timing — Mode 1							
Minimum Pulse Width: TPB	t_{TT}	5	100	150	100	150	ns
		10	50	75	—	—	
Minimum Setup Time: RSEL to Write	t_{RSW}	5	50	75	50	75	ns
		10	25	40	—	—	
Data to Write	t_{DW}	5	−100	−75	−100	−75	ns
		10	−50	−35	—	—	
Minimum Hold Time: RSEL after Write	t_{WRS}	5	50	75	50	75	ns
		10	25	40	—	—	
Data after Write	t_{WD}	5	75	125	75	125	ns
		10	40	60	—	—	

†Typical values are for T_A=25°C and nominal voltages.
*Maximum limits of minimum characteristics are the values above which all devices function.

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = −40 to +85°C, V_{DD} ±5%, t_r,t_f=20 ns, V_{IH}=0.7 V_{DD}, V_{IL}=0.3 V_{DD}, C_L=100 pF, see Fig. 6.

CHARACTERISTIC		V_{DD} (V)	LIMITS						UNITS
			CDP1854A			CDP1854AC			
			Min.	Typ.†	Max.*	Min.	Typ.†	Max.*	
CPU Interface — READ Timing — Mode 1									
Minimum Pulse Width: TPB	t_{TT}	5	—	100	150	—	100	150	ns
		10	—	50	75	—	—	—	
Minimum Setup Time: RSEL to TPB	t_{RST}	5	—	50	75	—	50	75	ns
		10	—	25	40	—	—	—	
Minimum Hold Time: RSEL after TPB	t_{TRS}	5	—	50	75	—	50	75	ns
		10	—	25	40	—	—	—	
Read to Data Access Time	t_{RDDA}	5	—	200	300	—	200	300	ns
		10	—	100	150	—	—	—	
Read to Data Valid Time	t_{RDV}	5	—	200	300	—	200	300	ns
		10	—	100	150	—	—	—	
RSEL to Data Valid Time	t_{RSDV}	5	—	150	225	—	150	225	ns
		10	—	75	125	—	—	—	
Hold Time: Data after Read	t_{RDH}	5	50	150	—	50	150	—	ns
		10	25	75	—	—	—	—	

†Typical values are for T_A=25°C and nominal voltages.
*Maximum limits of minimum characteristics are the values above which all devices function.

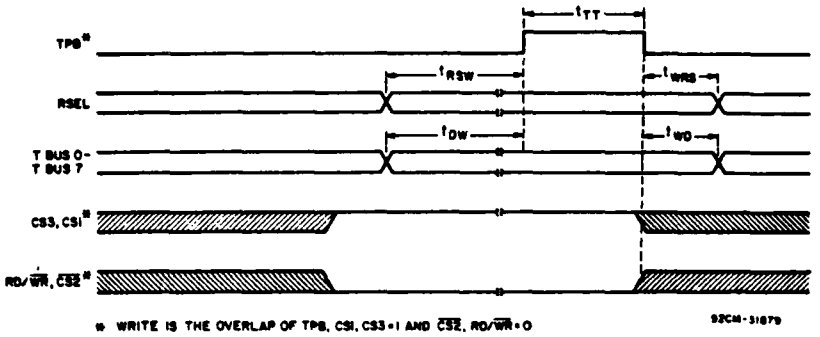


Fig. 5 - Mode 1 CPU interface (WRITE) timing diagram.

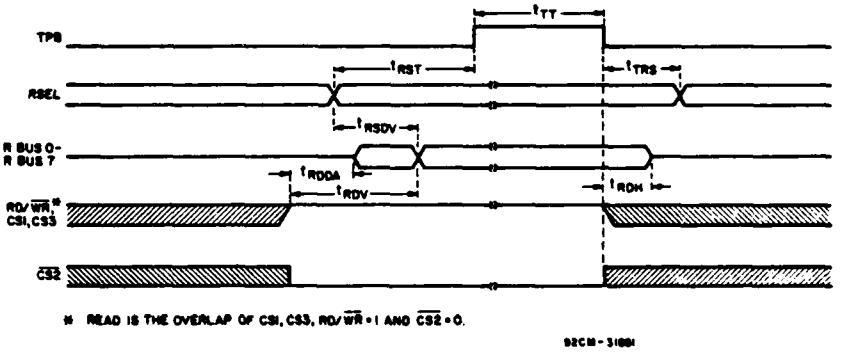


Fig. 6 - Mode 1 CPU interface (READ) timing diagram.

Mode Input Low (Mode = 0)

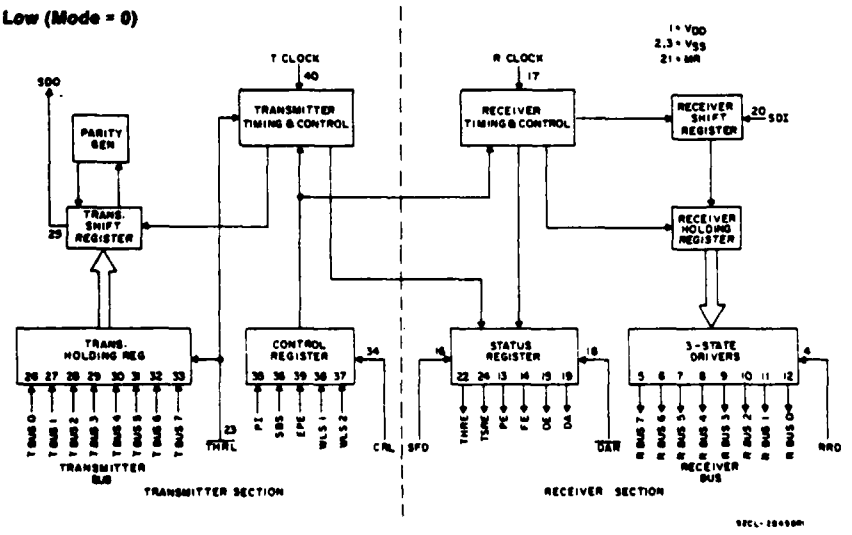


Fig. 7 - Mode 0 block diagram (industry standard compatible).

Functional Definitions
Standard Mode 0

SIGNAL: FUNCTION

VDD:
Positive supply voltag

MODE SELECT (MOD
A low-level voltage at
Operation.

VSS:
Ground.

RECEIVER REGISTEF
A high-level voltage a
Receiver Holding Reg

RECEIVER BUS (R B
Receiver parallel data

PARITY ERROR (PE)
A high-level voltage
received parity does
the EVEN PARITY E
updated each time a c
Holding Register. PE
bused together sinc
provided by the STA

FRAMING ERROR (F
A high-level voltag
received character
following the parity t
voltage. This output
transferred to the Re
a number of arrays c
disconnect capabili
DISCONNECT (SFD

OVERRUN ERROR
A high-level voltage
AVAILABLE (DA)
character was transf
OE lines from a nur
since an output dis
STATUS FLAG DIS

STATUS FLAG DIS
A high-level voltag
state output drivers
these status output

RECEIVER CLOCK
Clock input with a f
shift rate.

DATA AVAILABLE
A low-level voltag
flip-flop.

DATA AVAILABLE
A high-level voltag
character has been
Holding Register.

SERIAL DATA IN (
Serial data receive
register at a point
high-level voltage
received.

MASTER RESET (
A high-level volt
Holding Register,
and sets the serial

Functional Definitions for CDP1854A Terminals Standard Mode 0

SIGNAL: FUNCTION

V_{DD}:
Positive supply voltage.

MODE SELECT (MODE):
A low-level voltage at this input selects Standard Mode 0 Operation.

V_{SS}:
Ground.

RECEIVER REGISTER DISCONNECT (RRD):
A high-level voltage applied to this input disconnects the Receiver Holding Register from the Receiver Bus.

RECEIVER BUS (R BUS 7 - R BUS 0):
Receiver parallel data outputs.

PARITY ERROR (PE):
A high-level voltage at this output indicates that the received parity does not compare to that programmed by the EVEN PARITY ENABLE (EPE) control. This output is updated each time a character is transferred to the Receiver Holding Register. PE lines from a number of arrays can be bused together since an output disconnect capability is provided by the STATUS FLAG DISCONNECT (SFD) line.

FRAMING ERROR (FE):
A high-level voltage at this output indicates that the received character has no valid stop bit, i.e., the bit following the parity bit (if programmed) is not a high-level voltage. This output is updated each time a character is transferred to the Receiver Holding Register. FE lines from a number of arrays can be bused together since an output disconnect capability is provided by the STATUS FLAG DISCONNECT (SFD) line.

OVERRUN ERROR (OE):
A high-level voltage at this output indicates that the DATA AVAILABLE (DA) flag was not reset before the next character was transferred to the Receiver Holding Register. OE lines from a number of arrays can be bused together since an output disconnect capability is provided by the STATUS FLAG DISCONNECT (SFD) line.

STATUS FLAG DISCONNECT (SFD):
A high-level voltage applied to this input disables the 3-state output drivers for PE, FE, OE, DA, and THRE, allowing these status outputs to be bus connected.

RECEIVER CLOCK (RCLOCK):
Clock input with a frequency 16 times the desired receiver shift rate.

$\overline{\text{DATA AVAILABLE RESET}}$ ($\overline{\text{DAR}}$):
A low-level voltage applied to this input resets the DA flip-flop.

DATA AVAILABLE (DA):
A high-level voltage at this output indicates that an entire character has been received and transferred to the Receiver Holding Register.

SERIAL DATA IN (SDI):
Serial data received at this input enters the receiver shift register at a point determined by the character length. A high-level voltage must be present when data is not being received.

MASTER RESET (MR):
A high-level voltage at this input resets the Receiver Holding Register, Control Register, and Status Register, and sets the serial data output high.

TRANSMITTER HOLDING REGISTER EMPTY (THRE):
A high-level voltage at this output indicates that the Transmitter Holding Register has transferred its contents to the Transmitter Shift Register and may be reloaded with a new character.

$\overline{\text{TRANSMITTER HOLDING REGISTER LOAD}}$ ($\overline{\text{THRL}}$):
A low-level voltage applied to this input enters the character on the bus into the Transmitter Holding Register. Data is latched on the trailing edge of this signal.

TRANSMITTER SHIFT REGISTER EMPTY (TSRE):
A high-level voltage at this output indicates that the Transmitter Shift Register has completed serial transmission of a full character including stop bit(s). It remains at this level until the start of transmission of the next character.

SERIAL DATA OUTPUT (SDO):
The contents of the Transmitter Shift Register (start bit, data bits, parity bit, and stop (bit(s)) are serially shifted out on this output. When no character is being transmitted, a high-level is maintained. Start of transmission is defined as the transition of the start bit from a high-level to a low-level output voltage.

TRANSMITTER BUS (T BUS 0 - T BUS 7):
Transmitter parallel data inputs.

CONTROL REGISTER LOAD (CRL):
A high-level voltage at this input loads the Control Register with the control bits (PI, EPE, SBS, WLS1, WLS2). This line may be strobed or hardwired to a high-level input voltage.

PARITY INHIBIT (PI):
A high-level voltage at this input inhibits the parity generation and verification circuits and will clamp the PE output low. If parity is inhibited the stop bit(s) will immediately follow the last data bit on transmission.

STOP BIT SELECT (SBS):
This input selects the number of stop bits to be transmitted after the parity bit. A high-level selects two stop bits, a low-level selects one stop bit. Selection of two stop bits with five data bits programmed selects 1.5 stop bits.

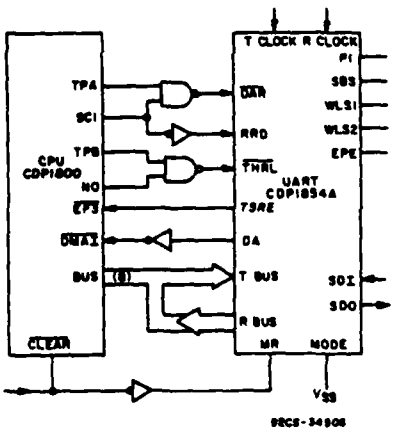


Fig. 8 - Mode 0 connection diagram.

WORD LENGTH SELECT 2 (WLS2):
WORD LENGTH SELECT 1 (WLS1):
These two inputs select the character length (exclusive of parity) as follows:

WLS2	WLS1	Word Length
Low	Low	5 Bits
Low	High	6 Bits
High	Low	7 Bits
High	High	8 Bits

EVEN PARITY ENABLE (EPE):
A high-level voltage at this input selects even parity to be generated by the transmitter and checked by the receiver. A low-level input selects odd parity.

TRANSMITTER CLOCK (TCLOCK):
Clock input with a frequency 16 times the desired transmitter shift rate.

Description of Standard Mode 0 Operation (Mode Input=V_{SS})

1. Initialization and Controls

The MASTER RESET (MR) input is pulsed, resetting the Control, Status, and Receiver Holding Registers and setting the SERIAL DATA OUTPUT (SDO) signal high. Timing is generated from the clock inputs, Transmitter Clock (TCLOCK) and Receiver Clock (RCLOCK), at a frequency equal to 16 times the serial data bit rate. When the receiver data input rate and the transmitter data output rate are the same, the TCLOCK and RCLOCK inputs may be connected together. The CONTROL REGISTER LOAD (CRL) input is pulsed to store the control inputs PARITY INHIBIT (PI), EVEN PARITY ENABLE (EPE), STOP BIT SELECT (SBS), and WORD LENGTH SELECTs (WLS1 and WLS2). These inputs may be hardwired to the proper voltage levels (V_{SS} or V_{DD}) instead of being dynamically set and CRL may be hardwired to V_{DD}. The CDP1854A is then ready for transmitter and/or receiver operation.

2. Transmitter Operation

For the transmitter timing diagram refer to Fig. 10. At the beginning of a typical transmitting sequence the Transmitter Holding Register is empty (THRE is HIGH). A character is transferred from the transmitter bus to the Transmitter holding Register by applying a low pulse to the $\overline{\text{TRANSMITTER HOLDING REGISTER LOAD}}$ ($\overline{\text{THRL}}$) input causing THRE to go low. If the Transmitter Shift Register is empty (TSRE is HIGH) and the clock is low, on the next high-to-low transition of the clock the character is loaded into the Transmitter Shift Register preceded by a start bit. Serial data transmission begins 1/2 clock period later with a start bit and 5-8 data bits followed by the parity bit (if programmed) and stop bit(s). The THRE output signal goes high 1/2 clock period later on the high-to-low transition of the clock. When THRE goes high, another character can be loaded into the Transmitter Holding Register for transmission beginning with a start bit immediately following the last stop bit of the previous character. This process is repeated until all characters have been transmitted. When transmission is complete, THRE and Transmitter Shift Register Empty (TSRE) will both be high. The format of serial data is shown in Fig. 12. Duration of each serial output data bit is determined by the transmitter clock frequency (f_{CLOCK}) and will be 16/f CLOCK.

3. Receiver Operation

The receive operation begins when a start bit is detected at the SERIAL DATA IN (SDI) input. After the detection of a high-to-low transition on the SDI line, a divide-by-16 counter is enabled and a valid start bit is verified by checking for a low-level input 7-1/2 receiver clock periods later. When a valid start bit has been verified, the following data bits, parity bit (if programmed), and stop bit(s) are shifted into the Receiver Shift Register at clock pulse 7-1/2 in each bit time. If programmed, the parity bit is checked, and receipt of a valid stop bit is verified. On count 7-1/2 of the first stop bit, the received data is loaded into the Receiver Holding Register. If the word length is less than 8 bits, zeros (low output voltage level) are loaded into the unused most significant bits. If DATA AVAILABLE (DA) has not been reset by the time the Receiver Holding Register is loaded, the OVERRUN ERROR (OE) signal is raised. One-half clock period later, the PARITY ERROR (PE) and FRAMING ERROR (FE) signals become valid for the character in the Receiver Holding Register. The DA signal is also raised at this time. The 3-state output drivers for DA, OE, PE and FE are enabled when STATUS FLAG DISCONNECT (SFD) is low. When RECEIVER REGISTER DISCONNECT (RRD) goes low, the receiver bus 3-state output drivers are enabled and data is available at the RECEIVER BUS (R BUS 0 - R BUS 7) outputs. Applying a negative pulse to the $\overline{\text{DATA AVAILABLE RESET}}$ ($\overline{\text{DAR}}$) resets DA. The preceding sequence of operation is repeated for each serial character received. A receiver timing diagram is shown in Fig. 11.

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = -40 to +85°C, V_{DD} ±5%, t_r,t_f=20 ns, V_{IH}=0.7 V_{DD}, V_{IL}=0.3 V_{DD}, C_L=100 pF, see Fig. 9.

CHARACTERISTIC		V_{DD} (V)	LIMITS				UNITS
			CDP1854A		CDP1854AC		
			Typ.†	Max.*	Typ.†	Max.*	
Interface Timing — Mode 0							
Minimum Pulse Width: CRL	t_{CRL}	5	50	150	50	150	ns
		10	40	100	—	—	
Minimum Pulse Width: MR	t_{MR}	5	300	400	300	400	ns
		10	150	200	—	—	
Minimum Setup Time: Control Word to CRL	t_{CWC}	5	20	50	20	50	ns
		10	0	40	—	—	
Minimum Hold Time: Control Word after CRL	t_{CCW}	5	40	60	40	60	ns
		10	20	30	—	—	
Propagation Delay Time: SFD High to SOD	t_{SFDH}	5	200	300	200	300	ns
		10	100	150	—	—	
SFD Low to SOD	t_{SFDL}	5	75	120	75	120	ns
		10	40	60	—	—	
RRD High to Receiver Register High Impedance	t_{RRDH}	5	200	300	200	300	ns
		10	100	150	—	—	
RRD Low to Receiver Register Active	t_{RRDL}	5	80	150	80	150	ns
		10	40	70	—	—	

†Typical values are for T_A=25°C and nominal voltages.
*Maximum limits of minimum characteristics are the values above which all devices function.

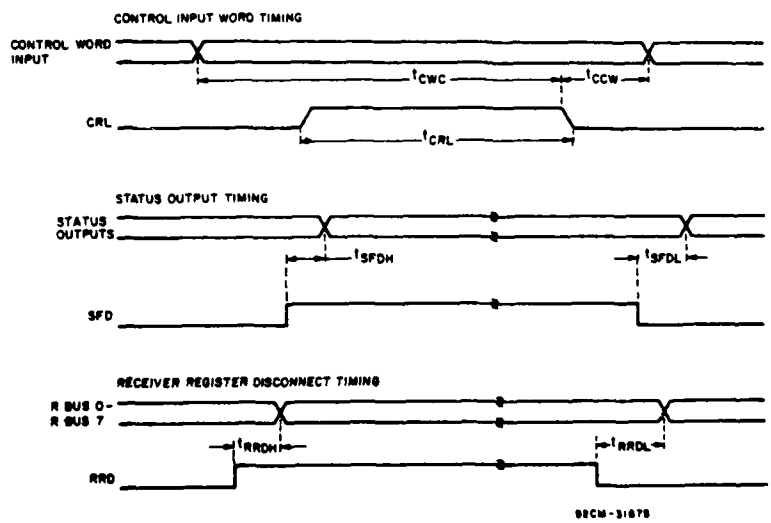
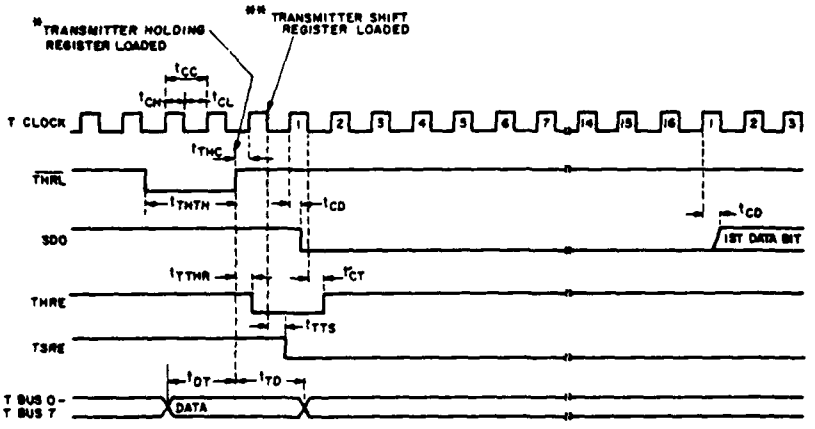


Fig. 9 - Mode 0 interface timing diagram.

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = −40 to +85°C, V_{DD} ±5%, t_r,t_f=20 ns, V_{IH}=0.7 V_{DD}, V_{IL}=0.3 V_{DD}, C_L=100 pF, see Fig. 10.

CHARACTERISTIC		V_{DD} (V)	LIMITS				UNITS
			CDP1854A		CDP1854AC		
			Typ.†	Max.*	Typ.†	Max.*	
Transmitter Timing — Mode 0							
Minimum Clock Period	t_{CC}	5	250	310	250	310	ns
		10	125	155	—	—	
Minimum Pulse Width:		5	100	125	100	125	ns
Clock Low Level	t_{CL}	10	75	100	—	—	
Clock High Level	t_{CH}	5	100	125	100	125	ns
		10	75	100	—	—	
$\overline{THRL}$	t_{THTH}	5	60	150	60	150	ns
		10	40	100	—	—	
Minimum Setup Time:		5	175	275	175	275	ns
$\overline{THRL}$ to Clock	t_{THC}	10	90	150	—	—	
Data to $\overline{THRL}$	t_{DT}	5	20	50	20	50	ns
		10	0	40	—	—	
Minimum Hold Time:		5	40	60	40	60	ns
Data after $\overline{THRL}$	t_{TD}	10	20	30	—	—	
Propagation Delay Time:		5	300	450	300	450	ns
Clock to Data Start Bit	t_{CD}	10	150	225	—	—	
Clock to THRE	t_{CT}	5	200	300	200	300	ns
		10	100	150	—	—	
$\overline{THRL}$ to THRE	t_{TTHR}	5	200	300	200	300	ns
		10	100	150	—	—	
Clock to TSRE	t_{TTS}	5	200	300	200	300	ns
		10	100	150	—	—	

†Typical values are for T_A=25°C and nominal voltages.
*Maximum limits of minimum characteristics are the values above which all devices function.



* THE HOLDING REGISTER IS LOADED ON THE TRAILING EDGE OF $\overline{THRL}$.
** THE TRANSMITTER SHIFT REGISTER, IF EMPTY, IS LOADED ON THE FIRST HIGH-TO-LOW TRANSITION OF THE CLOCK WHICH OCCURS AT LEAST 1/2 CLOCK PERIOD + t_{THC} AFTER THE TRAILING EDGE OF $\overline{THRL}$, AND TRANSMISSION OF A START BIT OCCURS 1/2 CLOCK PERIOD + t_{CD} LATER

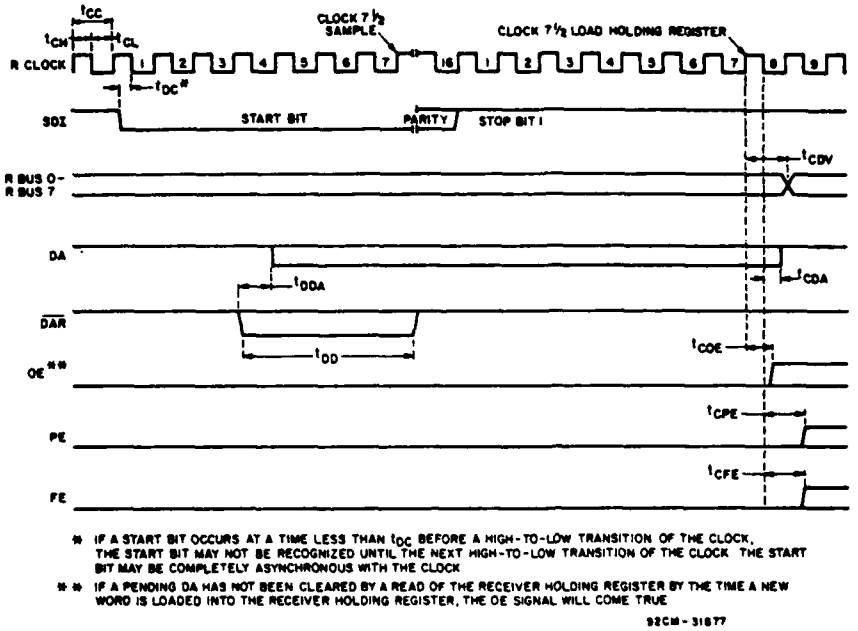
92CS-31875R1

Fig. 10 - Mode 0 transmitter timing diagram.

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = −40 to +85°C, V_{DD} ±5%, t_r,t_f=20 ns, V_{IH}=0.7 V_{DD}, V_{IL}=0.3 V_{DD}, C_L=100 pF, see Fig. 11.

CHARACTERISTIC		V_{DD} (V)	LIMITS				UNITS
			CDP1854A		CDP1854AC		
			Typ.†	Max.*	Typ.†	Max.*	
Receiver Timing — Mode 0							
Minimum Clock Period	t_{CC}	5	250	310	250	310	ns
		10	125	155	—	—	
Minimum Pulse Width: Clock Low Level	t_{CL}	5	100	125	100	125	ns
		10	75	100	—	—	
Clock High Level	t_{CH}	5	100	125	100	125	ns
		10	75	100	—	—	
$\overline{\text{DATA AVAILABLE}}$ RESET	t_{DD}	5	50	75	50	75	ns
		10	25	40	—	—	
Minimum Setup Time: Data Start Bit to Clock	t_{DC}	5	100	150	100	150	ns
		10	50	75	—	—	
Propagation Delay Time: $\overline{\text{DATA AVAILABLE}}$ RESET to Data Available	t_{DDA}	5	150	225	150	225	ns
		10	75	125	—	—	
Clock to Data Valid	t_{CDV}	5	225	325	225	325	ns
		10	110	175	—	—	
Clock to Data Available	t_{CDA}	5	225	325	225	325	ns
		10	110	175	—	—	
Clock to Overrun Error	t_{COE}	5	210	300	210	300	ns
		10	100	150	—	—	
Clock to Parity Error	t_{CPE}	5	240	375	240	375	ns
		10	120	175	—	—	
Clock to Framing Error	t_{CFE}	5	200	300	200	300	ns
		10	100	150	—	—	

†Typical values are for T_A=25°C and nominal voltages.
*Maximum limits of minimum characteristics are the values above which all devices function.

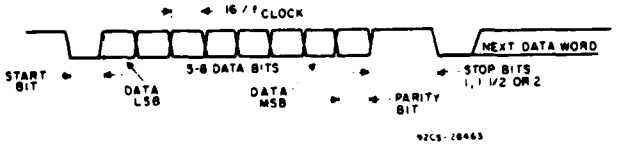


* IF A START BIT OCCURS AT A TIME LESS THAN t_{DC} BEFORE A HIGH-TO-LOW TRANSITION OF THE CLOCK, THE START BIT MAY NOT BE RECOGNIZED UNTIL THE NEXT HIGH-TO-LOW TRANSITION OF THE CLOCK. THE START BIT MAY BE COMPLETELY ASYNCHRONOUS WITH THE CLOCK

** IF A PENDING DA HAS NOT BEEN CLEARED BY A READ OF THE RECEIVER HOLDING REGISTER BY THE TIME A NEW WORD IS LOADED INTO THE RECEIVER HOLDING REGISTER, THE OE SIGNAL WILL COME TRUE

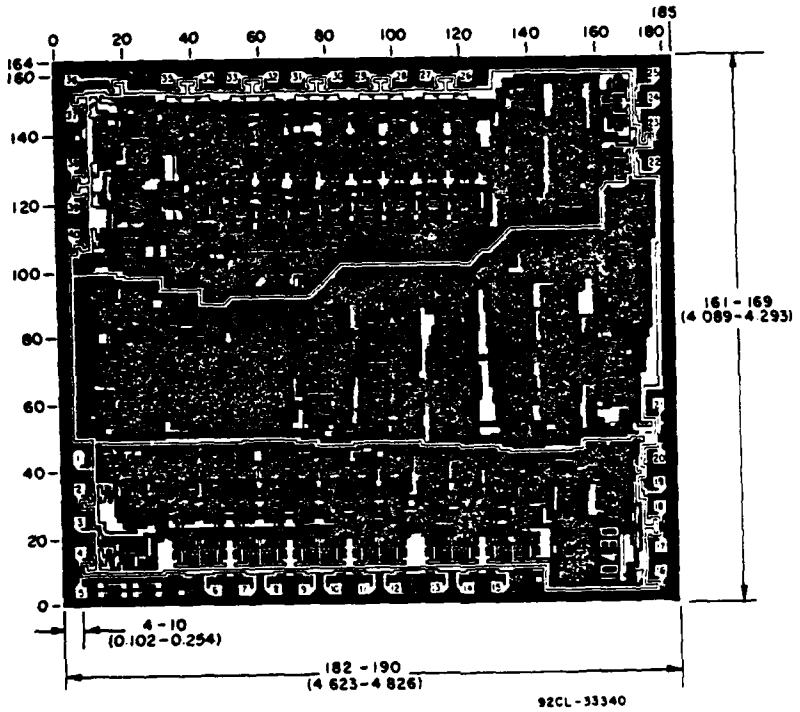
92CM-31877

Fig. 11 - Mode 0 receiver timing diagram.



92CS-28463

Fig. 12 - Serial data word format.



Dimensions and pad layout for CDP1854ACH.

Dimensions in parentheses are in millimeters and are derived from the basic inch dimensions as indicated. Grid graduations are in mils (10^{-3} inch).

The photographs and dimensions represent a chip when it is part of the wafer. When the wafer is cut into chips, the cleavage angles are 57° instead of 90° with respect to the face of the chip. Therefore, the isolated chip is actually 7 mils (0.17 mm) larger in both dimensions.

OPERATING AND HANDLING CONSIDERATIONS

1. **Handling**
   All inputs and outputs of RCA CMOS devices have a network for electrostatic protection during handling. Recommended handling practices for CMOS devices are described in ICAN-6525 "Guide to Better Handling and Operation of CMOS Integrated Circuits."

2. **Operating**

Operating Voltage

During operation near the maximum supply voltage limit, care should be taken to avoid or suppress power supply turn-on and turn-off transients, power supply ripple, or ground noise; any of these conditions must not cause V_{DD}—V_{SS} to exceed the absolute maximum rating.

Input Signals

To prevent damage to the input protection circuit, input signals should never be greater than V_{DD} nor less than V_{SS}. Input currents must not exceed 10 mA even when the power supply is off.

Unused Inputs

A connection must be provided at every input terminal. All unused input terminals must be connected to either V_{DD} or V_{SS}, whichever is appropriate.

Output Short Circuits

Shorting of outputs to V_{DD} or V_{SS} may damage CMOS devices by exceeding the maximum device dissipation.

d] Programmer's Manual

TABLE OF CONTENTS

1.0 INTRODUCTION

This manual is intended for anyone who desires a thorough knowledge of the internal control and data structures of the SIMSAM simulator. It is a companion to the program listings also included in this Appendix. The reader of this manual is assumed to be familiar with the BASIC language, and GW-Basic knowledge is especially helpful.

The program listings are fully commented, so some information will not be redundantly included here. This manual will explain the logic of sections of the program that are not clear in the listings, but will only make sense if reference is made to the listings, at the same time.

Certain sections in the program listings are given headings. The headings are easy to locate because they are enclosed in strings of asterisks. To find out information about a particular section of the program listings, match the heading to the same heading in this manual for the appropriate module. Program functions are grouped by the module they occur in. All variable names in this manual are shown in capital letters.

2.0 MAIN.BAS

The purpose of this module is to declare all arrays needed by the simulator, display the main menu, and merge other modules as needed with this module.

2.1 INITIALIZATION

This section takes care of the array declarations. Descriptions of the various arrays are included in the listing. The default drive (DRIVE$) is set to "B".

2.2 MAIN MENU

This section displays the SIMSAM MENU, also referred to as the main menu. This is the menu from which all other menus and functions are accessed.

2.3 CHAIN OTHER MODULES

After the user enters a selection number from 1 to 7, this section uses the CHAIN MERGE command to append the appropriate module to the end of the main module. If an invalid selection is made the SOUND command alerts the user.

All modules that are chained to the main module have line numbers which begin at 9000. All modules return control to the main module by a GOTO 5000 command.

3.0 INPOVR.BAS

This module loads the input waveform into memory, either from a disk file, or from the keyboard. It allows some simple editing of the file, and the ability to save the edited version.

Referring to the program listing, the first task is to display the INPUT WAVEFORM MENU. The menu mechanism is identical to that of the main module.

3.1 ENTER NUMBER OF INPUT LEVELS

MU% is set to be the number of input levels, and must be less than MMAX% (which was preset in the main module).

If the input waveform is to be obtained from a file (menu selection 1), then INPUTXX$ is set to the name of the file and a subroutine is called to read the data.

After the subroutine returns the Q bit is tested. This bit is used in a number of modules to indicate that the data read from the file is inconsistent with other information. For example, there is an error if MU%=5 and 6 input levels are read from the file, so Q would be set to 1.

3.2 SAVE INPUT WAVEFORM

Here, INPUTXX$ is set to the name of the file to write the input waveform data to.

3.3 ENTER INPUT WAVEFORM

This section is used to read the input waveform from the keyboard. NUMPU% is set to the number of distinct pieces in the input waveform. As the value and start times are entered from the keyboard, they are stored in the arrays UV%() and UT() respectively. A check is made to see that the value is an integer greater than zero, and not greater than MU%.

After all input data has been entered, it is echoed back to the screen for possible editing. If editing is desired, (X$="Y"), then the user enters piece number, value, and start time into J%, X%, and Y. If J% is not equal to 0, then UV%(J%) and UT(J%) are updated.

3.4 WRITE INPUT WAVEFORM

A file is opened on the default disk with the filename set in INPUTXX$ using the OPEN command. The input data is written in random format (refer to GW-Basic manual for details). Each record written is 6 bytes long: 2 bytes for

the input value, and 4 bytes for the start time of the value.

These pairs are written beginning with record 2. Record 1 is reserved for the number of input pieces (NUMPU%) and the number of input levels (MU%).

3.5 READ INPUT WAVEFORM

An input waveform is read from the default drive using the filename contained in INPUTXX$. The file has the same format as explained in the previous section.

SAVEDMU% is set to the number of input pieces read from the file. If this value is greater than NUMPU%, then Q is set to 1. Otherwise, UV%() and UT() arrays are initialized from data read from the input file.

4.0 DFGOVR.BAS

This module begins by displaying the DFG MENU and obtaining the user's selection.

4.1 ENTER NUMBER OF STATES

NS% is set to the number of states in the machine to be simulated. This value must be less than NMAX%, the maximum allowable number of states, preset in the main module.

4.2 SAVE DFGS

This section prompts for the filename to be stored in DFGXX$ so the DFGs can be written to disk.

4.3 ENTER DFGS

MAXP% had been set earlier to be equal to MAXD%, the maximum number of pieces allowed in any one DFG. The array PNM%%(,) is used to keep track of the number of pieces for a particular input level and machine state. ID% is the index for the input level, and JD% is the index for the state.

Each piece of a DFG is stored in two arrays: DFGV%() holds the value, and DFGT() holds the start time. The value must be in range, and successive start times must be monotonically increasing. J% is the index for the piece number of a DFG.

If any changes are desired after the arrays DFGV%() and DFGT() have been displayed, the user inputs the piece number, value, and start time into J%, X%, and Y. If J% is not equal to zero, the piece of the DFG is updated.

4.4 WRITE DFGS

The filename in DFGXX$ is opened as a random file on the default drive. The record length is 6 bytes. Field 1 is two bytes, and field 2 is four bytes.

For record 1, the first two bytes contain the number of pieces in the first DFG, and the last 4 bytes contain the number of input levels (MU%) as the integer part, and the number of machine states (NS%) as the fractional part.

Starting with the second record, ID% keeps track of the current piece number, I2% tracks the input level, and J2% tracks the state level. First the number of pieces for the current DFG are written in field 1, then the values of DFGV%() and DFGT() for the current DFG are written in field 1 and 2 respectively of subsequent records. This is repeated for each of the DFGs.

4.5 READ DFGS

DFGs are read from the filename in DFGXX$. The file has the format explained in the previous section. SAVEDMU% is set to the number of input levels read from the file. SAVEDNS% is set to the number of states read from the file. If either of these quantities differ from the quantities in memory, an errror is reported by setting Q=1.

4.6 ECHO DFGS

The values of DFG(I1%,J1%) are listed on the screen, where I1% is the input level, and J1% is the state. J% stores the current piece number. The listing gives J%, DFGV%(J%, I1%, J1%), and DFGT(J%, I1%, J1%). If the user wants to change any piece of the current DFG, the piece # is entered into J%, the value into X%, and the start time into Y.

5.0 LGFOVR.BAS

This module begins by displaying the LF MENU, and accepting the user selection.

5.1 SAVE LF

If the Logic Function is to be saved, the name of the file is stored in LFXX$.

5.2 ENTER LF

This section allows the logic function to be input from the keyboard. YMAX% holds the number of output levels. The array LF%() stores an output value for each of the MU% possible input levels, and each of the NS% possible states.

Edits to the LF for the current input level are possible by specifying the state (I%) and the revised output level (Y%).

5.3 WRITE LF

The LF data is written into the random file with the filename in LFXX$. Each record is two bytes long.

The first record contains the number of states, the second contains the number of number of input levels, and the third contains the number of output levels. The rest of the records contain the output values for a given input level and state.

5.4 READ LF

The filename in LFXX$ is read from the default drive with the format detailed in the previous section. SAVEDNS% is set to the number of states read from the file. SAVEDMU% is set to the number of input levels read from the file. If either of these differ from the values currently in memory and error occurs, and Q is set to 1. Otherwise, Q=0.

6.0 RSMOVR.BAS

This module takes the input waveform, DFGs and LF data comprising the model of the machine to be simulated, to execute the simulation.

6.1 ENTER MAXIMUM EVENTS

The maximum number of events to consider is stored in EVENTS%. This value must not be greater than MAXY%, which is preset in the main menu.

6.2 ENTER PROCESSING TIME

A finite processing time can be stored in TPROC. Any changes that occur during this processing time are ignored. This point is clarified in the next section.

Before the simulation can proceed, the initial state must be specified, and is stored in XO%. IND% is set to contain the actual number of events in the simulation.

If the results of the simulation run are to be saved, the filename is entered into RESXX$.

6.3 RUN SIMULATION

This section is probably the most complex of the program since it encompasses the actual simulation of the machine. The program flow will be explained here with reference to the variables contained in the program listing.

First, key variables are initialized. TINF is set to 32000. TINF is a large time period, approximating infinity. INDEX% keeps track of the number of pieces in the output signal. IX% holds the current DFG piece number. IL% holds the input signal piece number. UI% contains the current input level. YL% contains the previous state (initially equal to zero).

XJ% contains the current state, and is based on the current DFG piece number, the current input level, and the last state. The start time of the first output piece, [illegible], is zero. The value of the first output piece, [illegible], is the value of LF at the current input and state levels. The input level at this time is stored in the first element of UPES%(). The state at this time is stored in the first element of XPES%().

The start time of the next pieces of the input signal, IL, and the current DFG, T, are obtained. The DFG piece number, IX% is incremented. On, time the input signal changes during the processing time, CD% is set to 1, and a change in input signal changes during processing is set to 1.

If the input signal changes during processing time (meaning that the change will not be acknowledged by the machine), get the time to the next change and see if it is greater than the processing time, TPROC. If it is, one now knows what the input level will be at the conclusion of the processing time.

Based on the values of TU and TX, either the state changes first, or the input does. If the state does, then XJ% is set equal to the next piece of the current DFG. The arrays URES%(), XRES%(), YV%(), and YT() are updated with the current information. Increment INDEX% and continue to loop as it has not exceeded EVENTS%, the maximum number of events.

If the input changes before the state, get the next input value. Set XL% equal to XJ% (save the old state). Set the new state equal to the first state value of the new DFG. Update arrays URES%(), XRES%(), YV%() and YT(). Increment INDEX% and check that it is less than EVENTS%, the maximum number of events.

Update TU and TX and continue until the maximum number of events has been exceeded, or the input stops changing, and current DFG stops changing.

6.4 SAVE SIMULATION RESULTS

This section saves the simulation results in the filename contained in RESXX$. The data is written into a random file with a record length of 10 bytes.

The first field of the record is 2 bytes, the second is 2 bytes, the third is 2 bytes, and the fourth field is 4 bytes.

In the first record, the last two characters of the input filename are stored in the first field. The last two characters of the DFG filename are stored in the second field. The last two characters of the LP filename are stored in the third field. The number of points [illegible] stored in the fourth field.

In the second record, field 1 contains the number of input levels, XMAX%. Field 2 contains the number of states, MXJ%. Field 3 contains the number of events, EVENTS%. Field 4 contains the initial state, XJ%.

In record 3, field 1 contains a code number indicating whether an error occurred during the processing [illegible] 2 contains the processing time, TPROC. Field 3 and 4 contain [illegible] client.

Starting with record 4, the rest of the records in the results file contain the results simulation. Field 1 holds the output value, YV%(). Field 2 holds the input value, URES%(). Field 3 contains the state value, XRES%(). Field 4 contains the start time for the output value, YT().

7.0 PLTOVR.BAS

This module starts by displaying the PLOT MENU, and prompting for the user's selection of a plotting function.

7.1 PLOT OUTPUT

IND% was set in the RSMOVR module to the number of pieces in the output signal. The array FPLT%() is initialized with the values of the output signal from YV%(). NAM% is used to hold the code for the type of plot requested. Here, NAM% is set to 1. LEVELS% is set to YMAX%, the number of levels in the output signal.

7.2 PLOT INPUT

The array FPLT%() is initialized with the values of the input signal from URES%(). Because of this, the plotted input signal is not the original one entered by the user, but the one seen by the simulated machine, accounting for the processing time. NAM% is set to 2. LEVELS% is set to the number of input levels. CODE% is set to CD%, indicating if an input transition occurred during the processing time.

7.3 PLOT STATE

XRES%() is loaded into FPLT%(). NAM% is set to 3. LEVELS% is set to the number of states.

7.4 PLOT DFGS

If DFGs are to be plotted, the time scale for the DFGs must be adjusted to correspond with the output function.

The input, DFG, and LF filenames are saved in SPEC$$. TIMEPLT() array is initialized with the transition times from YT(), the output function start times.

7.5 PLOT REQUESTED FUNCTION

The array VALL%() is filled with the levels of the signal to be plotted from FPLT%(), and TIM() is filled with the start time of each piece of the signal to be plotted from TIMEPLT(). NP% is initialized with the number of pieces. CAPTION% is set to the code for type type of plot requested. STAR% is set to a 1 if CD% was set to 1 in the simulation (meaning that the simulation was affected by the processing time).

7.6 PLOTTER FUNCTION

The screen is set to high-resolution graphics mode. The time of last piece of the output signal is set to XMAX!. The appropriate title is printed at the top of the screen

based on the value of CAPTION%. The Y-axis is drawn and labeled using N%, the number of levels in the signal to be plotted.

Next, the x-axis is plotted using 10 equal subintervals of TLAST (the last element of TIM()). Finally, the waveform is plotted using the values from the arrays TIM() and VALL(), scaled to fill the screen.

The INKEY$ command is used to allow the Esc key to pass control back to the PLOT MENU or, in the case of DFGs, plot the next DFG.

7.7 LOAD RESULTS FILE

The results file whose filename is contained in RESX$ is read from the default drive. For details about the format of this file, refer to the SAVE SIMULATION RESULTS section of the RSMOVR module.

7.8 ADJUST DFG TIME SCALE

To be able to plot the DFGs correctly, the time scale for the DFGs must be expanded to be equal to the time scale for the output function. VALL% () is loaded with the values of the DFGV%() array. TIM() is loaded with the values of the DFGT() array. N% is set equal to the number of states. CAPTION% is set to 4.

e] Program Listings

```
1000    REM                          MAIN.BAS MODULE
1001    REM ************************************************************
1002    REM
1020    REM      This program simulates the behavior of simple asynchronous
1040    REM      machines (SAMs).  The simulation proceeds in an event-driven
1060    REM      fashion and provides for graphical output as well as file-
1080    REM      oriented input and output.  It is assumed that the user
1100    REM      is familiar with the contents and constructs of the paper
1120    REM      "Realization of Asynchronous Finite-State Machines," by
1140    REM      Johnson and Kaliski.  For initial storage allocation,
1160    REM      values for the following key variables will be preset:
1180    REM
1200    REM       NMAX%:  The maximum number of distinct states allowed in a SAM
1220    REM       MMAX%:  The maximum number of distinct input levels for the SAM
1240    REM       MAXU%:  The maximum number of pieces allowed in an input signal
1260    REM       MAXD%:  The maximum number of pieces allowed in any DFG
1280    REM       MAXY%:  The maximum number of pieces present in the generated
1300    REM               output signal. To be consistent with the assumed
1320    REM               SAM model, MAXY% = MAXU% * MAXD%
1340    REM
1360    REM      They will be preset to the following values, values which
1380    REM      can be changed at any time, of course.
1400    REM
1420    REM
1440              NMAX% = 9: MMAX% = 8
1460              MAXU% = 30 : MAXD% = 8
1480              MAXY% = MAXU% * MAXD%
1500    REM
1520    REM      To facilitate the writing of a common subroutine
1540    REM      for inputting both the SAM input signal and its DFGs we
1560    REM      set the variable MAX% to the maximum of MAXU% and MAXD%.
1580    REM
1600              IF MAXU% > MAXD% THEN MAX% = MAXU% ELSE MAX% = MAXD%
1620    REM
1640    REM
1660    REM      The simulator uses arrays to represent signals, as well
1680    REM      as other key entities.  The following arrays comprise the
1700    REM      simulator's data base:
1720    REM
1740    REM       UV%(MAXU%): The values of the pieces of the input function
1760    REM             in temporal sequence,
1780    REM       UT(MAXU%): The times that the distinct pieces of the input
1800    REM             function begin, starting with time = 0,
1820    REM       DFGV%(MAXD%,MMAX%,NMAX%): DFGV%(.,i,j) records the values
1840    REM             of the pieces of the i,j th DFG in temporal sequence,
1860    REM       DFGT(MAXD%,MMAX%,NMAX%): DFGT%(.,i,j) records the times
1880    REM             that the pieces in the i,jth DFG begin, starting
1900    REM             with time = 0,
1920    REM       YV%(MAXY%): The values of the pieces of the generated
1940    REM             output function in temporal sequence,
1960    REM       YT(MAXY%): The times that the distinct pieces of the
1980    REM             output function begin, starting at t=0,
2100    REM       NUMAX%(NMAX%,MMAX%): This array contains at entry i,j
2120    REM             the number of pieces present in the i,jth DFG.
```

```
2140    REM       LF%(MMAX%,NMAX%):  This array contains at entry i,j the
2160    REM              logical function output value produced for the
2180    REM              ith input and jth state value,
2200    REM       URES%(MAXY%):  The temporal values of the SAM input
2220    REM              at each system "event" time -- see below --,
2240    REM       XRES%(MAXY%): The temporal values of the SAM state
2260    REM              at each system "event" time -- see below --,
2280    REM     FPLT%(50): The simulator uses this array to save the values
2300    REM              for the asynchronous signals to be plotted later.
2320    REM     TIMEPLT(50): The start times associated with each function
2340    REM              to be plotted are saved in this array.
2420    REM     NAM%: This variable contains the identity of the functions
2440    REM              to be plotted. The i.d. is the same as the number used
2460    REM              to select the function in the plot menu.
2480    REM     SPEC$: This variable contains the specifics of run from which
2500    REM              plot is requested. Such information as the input
2520    REM              function, the DFG and the LF used, are saved in this
2540    REM              variable, provided these have been saved using the
2560    REM              appropriate file name formats given in this program.
2580    REM     CODE%:  The variable is 1, if due to a
2600    REM              non-zero processing time, an input event occurs while a
2620    REM              previous event is still being processed, during the
2640    REM              run for which plots are requested.
2780    REM     VALL%(50): This array is used by the plotter subroutine to
2800    REM              store the values of the different pieces in the
2820    REM              function to be plotted.
2840    REM     TIM(50): This array is used by the plotter subroutine, and
2860    REM              holds the start times of the pieces in the function
2880    REM              to be plotted.
2900    REM     LEVELS%: The plotter subroutine requires the largest
2920    REM              ordinate value to appear in each plot. This is saved
2940    REM              in LEVELS%
2960    REM
```

```
>2965     REM ********** INITIALIZATION **********
2966     REM
2980     REM      The arrays are dimensioned as above:
3000     REM
3020               DIM UV%(MAXU%),UT(MAXU%),DFGV%(MAXD%,MMAX%,NMAX%),
DFGT(MAXD%,MMAX%,NMAX%),YV%(MAXY%),YT(MAXY%)
3040               DIM KXMAX%(MMAX%,NMAX%),LF%(MMAX%,NMAX%),URES%(MAXY%),
XRES%(MAXY%)
3060               DIM FPLT%(50), TIMEPLT(50)
3100               DIM VALL%(100), TIM(100)
3110 DRIVE$="B"          'default drive for files
3114     REM
3115     REM ********** MAIN MENU **********
3116     REM
3120               CLS
3140 LOCATE 12,34 : PRINT "S I M S A M"
3160 LOCATE 24,19 : PRINT "copyright (c) 1985 Northeastern University";
3180 FOR I = 1 TO 1000 : NEXT I
3200 CLS : LOCATE 2,30 : PRINT "S I M S A M   M E N U"
3220 LOCATE 7,25 : PRINT "1 -- Enter Input waveform"
3240 LOCATE 9,25 : PRINT "2 -- Enter Digital Function Generators"
3260 LOCATE 11,25 : PRINT "3 -- Enter Logic Function Generator"
3280 LOCATE 13,25 : PRINT "4 -- Run simulation"
3300 LOCATE 15,25 : PRINT "5 -- Plot results "
3320 LOCATE 17,25 : PRINT "6 -- Set default drive"
3330 LOCATE 19,25 : PRINT "7 -- Exit"
3340 LOCATE 23,20 : PRINT SPC(40)
3360 LOCATE 23,20 : INPUT "Enter selection by number ": SELECT%
3365     REM
3366     REM ********** CHAIN OTHER MODULES **********
3367     REM
3370 IF SELECT%=SELECT.OLD% GOTO 9000
3380 ON SELECT% GOTO 3419,3439,3459,3479,3499,3410,3520
3400 SOUND 200,2 : GOTO 3340
3410 CLS : INPUT "Default drive ";DRIVE$ : GOTO 5000
3419 OVERLAY$=DRIVE$+":INFOVR" : GOTO 3500
3439 OVERLAY$=DRIVE$+":DFGOVR" : GOTO 3500
3459 OVERLAY$=DRIVE$+":LGFOVR" : GOTO 3500
3479 OVERLAY$=DRIVE$+":RSMOVR" : GOTO 3500
3499 OVERLAY$=DRIVE$+":PLTOVR"
3500 CHAIN MERGE OVERLAY$,9000,ALL,DELETE 9000-65529
3520 CLS : END
9000 SELECT.OLD%=SELECT% : GOTO 3200
9999 REM
10000 REM
65529 REM
```

```
9000    REM                          INPOVR.BAS MODULE
9001    REM ************************************************************
9002    REM
10000 CLS : LOCATE 2,22 : PRINT "I N P U T   W A V E F O R M   M E N U"
10020 LOCATE 11,25 : PRINT "1 -- Load input waveform from file"
10040 LOCATE 13,25 : PRINT "2 -- Enter input waveform from keyboard"
10060 LOCATE 15,25 : PRINT "3 -- Modify input waveform in memory"
10080 LOCATE 17,25 : PRINT "4 -- Exit to main menu"
10100 LOCATE 23,20 : PRINT SPC(40)
10120 LOCATE 23,20 : INPUT "Enter selection by number ": SUBSEL%
10140 ON SUBSEL% GOTO 10180,10180,11740,5000
10160 SOUND 200,2 : GOTO 10100
10165   REM
10166   REM ********** ENTER NUMBER OF INPUT LEVELS *********
10167   REM
10180    REM      We begin by obtaining the model parameters that serve
10200    REM      to define the specific SAM being simulated.
10220             CLS : PRINT "Enter M (<="MMAX%"), the number of input levels"
10240             PRINT
10260             INPUT "M = ",MU%
10280             IF MU% > MMAX% GOTO 10320
10300             IF MU% > 0 GOTO 10360
10320             PRINT "M - the number of input levels - out of range."
10340             GOTO 10260
10360 ON SUBSEL% GOTO 10440,10600
10380   REM
10400   REM      Input signal is in a random file.
10420   REM
10440             PRINT
10460             INPUT "File name for input  ";INFUF$
10480             GOSUB 12560              'call SUBROUTINE RESLINF
10500   REM      Test Q "bit" to see if input read is compatible with the
10520   REM      current SAM.
10540             IF Q=1 GOTO 10600
10560             GOTO 10000                  'skip call to READSG
10580   REM
10600   REM      Input signal is to be obtained from the terminal.
10620   REM      The variable MAXP% is a dummy variable used to allow use of the
10640   REM      same routine READSG for both the input function and the CFS.
10660   REM
10680             MAXP%=MAXU%: PRINT : PRINT "Enter input signal"
10700             GOTO 11280               'call SUBROUTINE READSG
10720   REM
10740   REM ********** SAVE INPUT WAVEFORM ********
10741   REM
10760   REM      The input function just entered may be saved as a random access
10780   REM      file organized as follows: record #1 contains the number of
10800   REM      records in the file (pieces of the signal) and the two arrays
10820   REM      which define the function are saved starting at record #2.
10840   REM      Each record of the file consists of two fields: a four byte
10860   REM      field which holds the signal level and a four byte field
10880   REM      which holds the start time of the level in the first field.
10900   REM      Note that if the input comes from the screen or keyboard
10920   REM      and not saved, the identity of this input function will be
```

```
10940    REM      saved along with the results from any run which uses this
10960    REM      input. The user must be alerted to this fact.
10980              PRINT "Note that if this input function is not saved, and"
11000              PRINT "the results of a run using this input are saved, such"
11020              PRINT "a result file will NOT contain information on the"
11040              PRINT "input function which was used." : PRINT
11060    REM
11080              INPUT "Save input? Yes(Y) or No (CR) ";Z$
11100              IF Z$="Y" GOTO 11140
11120              IF Z$<>"y" GOTO 10000
11140    REM      Input is to be saved.
11160              PRINT "Enter file name under which input is to be saved"
11180              PRINT "in the form 'INPUTxx', where xx are alphanumeric"
11200              INPUT "characters. ";INPUTXX$
11220              GOSUB 12120                          'call SUBROUTINE WRITEINF
11230               GOTO 10000
11240    REM
11245    REM ********** ENTER INPUT WAVEFORM **********
11246    REM
11260     REM      ****SUBROUTINE READSG****
11280     REM      This subroutine is used to enter input function
11300     REM
11320               PRINT "Enter number of pieces, <="MAXP%
11340               INPUT NUMPU%
11360               IF NUMPU%<=MAXP% THEN GOTO 11400 ELSE PRINT "# pieces must
not exceed"MAXP%
11380               GOTO 11340
11400               PRINT "Enter waveform in the format: value and start time"
11420               FOR J%=1 TO NUMPU%
11440               PRINT "piece #"J%;
11460               INPUT UV%(J%),UT(J%)
11480     REM      Check to make sure that 1<=UV%(J%)<=MU%
11500               IF UV%(J%) < 1 GOTO 11540
11520               IF UV%(J%) <= MU% GOTO 11580
11540               PRINT "Input level out of range."
11560               GOTO 11440                           're-enter the piece
11580               IF J%=1 GOTO 11720
11600               IF UT(J%)>UT(J%-1) THEN GOTO 11640 ELSE PRINT "time must
be monotone increasing":GOTO 11440
11620     REM      Check # of distinct levels
11640               FOR L%=J%-1 TO 1 STEP-1
11660               IF UV%(J%)=UV%(L%) GOTO 11700
11680               NUMLU%=NUMLU%+1
11700               NEXT L%
11720               NEXT J%
11740     REM      Echo waveform just entered
11760               PRINT:PRINT "The waveform is as follows:"
11780               PRINT "Piece #"TAB(10)"Value"TAB(20)"Start time"
11800               PRINT "_______"TAB(10)"_____"TAB(20)"__________"
11820               FOR J%=1 TO NUMPU%
11840               PRINT TAB(3)J%;TAB(10)UV%(J%);TAB(20)UT(J%)
11860               NEXT J%
11880               INPUT "Any changes (yes or [CR]) ";Z$
11900               IF Z$="y" AND Z$<>"Y" GOTO 11
```

```
11920             PRINT "Enter changes in the format: piece#, value and start
time"
11940             PRINT "To end enter piece # of 0"
11960             INPUT J%,X%,Y
11980             IF J%=0 GOTO 11760
12000             IF J>NUMPU% GOTO 12060
12020             UV%(J%)=X%: UT(J%)=Y
12040             GOTO 11960
12060             PRINT "Piece # must not exceed "MA%P%
12080             GOTO 11960
12090   REM
12091   REM ********** WRITE INPUT WAVEFORM **********
12092   REM
12100    REM     ****SUBROUTINE WRITEINF****
12120   REM      This subroutine is used for saving the input signal just enter-
12140   REM      ed. The input is saved in the file whose name is specified at
12160   REM      prompt time.
12180   REM
12200             IF DRIVE$="B" THEN FILE$="B:"+INPUTXX$ ELSE FILE$=INPUTXX$
12220             IF DRIVE$="b" THEN FILE$="B:"+INPUTXX$
12240             OPEN FILE$ AS #1 LEN=6
12260             FIELD #1,2 AS V$, 4 AS T$
12280             FOR JU%=2 TO NUMPU%+1
12300             LSET V$=MKI$(UV%(JU%-1))
12320             LSET T$=MKS$(UT(JU%-1))
12340             PUT #1,JU%
12360             NEXT JU%
12380             LSET V$=MKI$(NUMPU%)
12400             LSET T$ = MKI$(MU%)
12420             PUT #1,1
12440             CLOSE #1
12460             RETURN
12480   REM
12500   REM
12501   REM ********** READ INPUT WAVEFORM **********
12502   REM
12520   REM     ****SUBROUTINE READINF****
12540   REM
12560   REM      This subroutine is used to read back an input signal that has
12580   REM      been previously entered and saved in a random access file.
12600   REM
12620             IF DRIVE$="B" THEN FILE$="B:"+INPUTXX$ ELSE FILE$=INPUTXX$
12640             IF DRIVE$="b" THEN FILE$="B:"+INPUTXX$
12660             OPEN FILE$ AS #1 LEN=6
12680             FIELD #1, 2 AS V$, 4 AS T$
12700             GET #1,1
12720             NUMPU%=CVI(V$)
12740             SAVEDMU% = CVI(T$)
12760             IF SAVEDMU% = MU% GOTO 12900
12780             IF SAVEDMU% < MU% GOTO 12840
12800             PRINT "Input function has more levels than allowed"
12820             PRINT "in current SAM."
12840             Q = 1: GOTO 13040
12860             PRINT "Input function has fewer levels than allowed"
```

```
12880              PRINT " in current SAM. It will therefore be used."
12900              FOR JU% = 2 TO NUMPU%+1
12920              GET #1,JU%
12940              JU1% = JU% -1
12960              UV%(JU1%)=CVI(V$)
12980              UT(JU1%)=CVS(T$)
13000              NEXT JU%
13020              Q = 0
13040              CLOSE #1
13060              RETURN
13080    REM
55529 ' THIS MARKS THE END OF THE INPUT MODULE
```

```
9000     REM                                    DFGOVR.BAS MODULE
9001     REM ************************************************************
9002     REM
10000 CLS : LOCATE 2,37 : PRINT "D F G   M E N U"
10020 LOCATE 11,25 : PRINT "1 -- Load DFG from file"
10040 LOCATE 13,25 : PRINT "2 -- Enter DFG from keyboard"
10060 LOCATE 15,25 : PRINT "3 -- Modify DFG in memory"
10080 LOCATE 17,25 : PRINT "4 -- Exit to main menu"
10100 LOCATE 23,20 : PRINT SPC(40)
10120 LOCATE 23,20 : INPUT "Enter selection by number ";SUBSEL%
10140 ON SUBSEL% GOTO 10180,10180,10520,5000
10160 SOUND 200,2 : GOTO 10100
10170   REM
10171   REM ********** ENTER NUMBER OF STATES **********
10172   REM
10180    REM      We begin by obtaining the model parameters that serve
10200    REM      to define the specific SAM being simulated.
10220    REM
10240              CLS : PRINT "Enter N (<="NMAX%"), the number of states "
10260              PRINT
10280              INPUT "N = ",NS%
10300              IF NS% > NMAX% GOTO 10340
10320              IF NS% > 0 GOTO 10380
10340              PRINT "N - the number of states - out of range."
10360              GOTO 10280
10380 ON SUBSEL% GOTO 10400,10720
10400              INPUT "File name from which DFGs are to be read. ",DFGX$
10420              GOSUB 12660                    'call SUBROUTINE READDFG
10440   REM      Test Q "bit" to see if DFG read from file is compatible
10460   REM      with the current SAM.
10480              IF Q=1 GOTO 10720 ELSE 10000   'prompt for source of DFGs.
10500              PRINT
10520              FOR I1% = 1 TO MU%
10540              FOR J1% = 1 TO NS%
10580              GOSUB 13620                   'call SUBROUTINE ECHODFG
10600              NEXT J1%
10620              NEXT I1%
10640              GOTO 10980
10660    REM     Enter DFGs
10680    REM
10720             NUMLD%=1
10740              MAXP%=MAXD%
10760              FOR ID%=1 TO MU%
10780              FOR JD%=1 TO NS%
10800              PRINT "Enter DFG for input level"ID%"and state"JD%.
10820              GOSUB 11180
10840    REM
10860    REM      Update maximum # of pieces in a DFG
10880    REM
10900              IF NUM1%>NUMLD% THEN NUMLD%=NUM1%
10920              NEXT JD%
10940              NEXT ID%
10960    REM
10980   REM
```

```
10971    REM ********** SAVE DFGS **********
10972    REM
10980             PRINT
11000             INPUT "Are DFGs to be saved? (Y)es or No ":Z$
11020             IF Z$="y" GOTO 11080
11040             IF Z$<>"Y" GOTO 11120
11060             PRINT
11080             INPUT "Enter file name under which DFG is to be saved ";DFGXX$
11100             GOSUB 12020
11120 GOTO 10000
11130    REM
11131    REM ********** ENTER DFGS **********
11132    REM
11140             PRINT
11160     REM     ****SUBROUTINE READSG****
11180     REM     This subroutine is used to enter DFGs.
11200              PRINT "Enter number of pieces, (<="MAXP%")"
11220              INPUT KXMAX%(ID%,JD%)
11240              IF KXMAX%(ID%,JD%)<=MAXP% THEN GOTO 11280 ELSE PRINT "# pieces
 must
not exceed"MAXP%
11260              GOTO 11220
11280              PRINT "Enter waveform in the format: value and start time"
11300              FOR J%=1 TO KXMAX%(ID%,JD%)
11320              PRINT "piece #"J%;
11340              INPUT DFGV%(J%,ID%,JD%),DFGT(J%,ID%,JD%)
11360     REM     Check to make sure that 1<=dfgv%(J%,id%,jd%)<=NS%
11380              IF DFGV%(J%,ID%,JD%) < 1 GOTO 11420
11400              IF DFGV%(J%,ID%,JD%) <= NS% GOTO 11460
11420              PRINT "State level out of range."
11440              GOTO 11320                          're-enter the piece
11460              IF J%=1 GOTO 11600
11480              IF DFGT(J%,ID%,JD%) > DFGT(J%-1,ID%,JD%) GOTO 11520 ELSE PRINT
"Time must be monotone increasing":GOTO 11320
11500     REM     Check # of distinct levels
11520              FOR L%=J%-1 TO 1 STEP-1
11540              IF DFGV%(J%,ID%,JD%)=DFGV%(L%,ID%,JD%) GOTO 11580
11560            NUM1% = NUM1%+1
11580              NEXT L%
11600              NEXT J%
11620     REM     Echo waveform just entered
11640              PRINT:PRINT "The waveform just entered is as follows:"
11660              PRINT "Piece #"TAB(10)"Value"TAB(20)"Start time"
11680              PRINT "_______"TAB(10)"_____"TAB(20)"__________"
11700              FOR J%=1 TO KXMAX%(ID%,JD%)
11720              PRINT TAB(3)J%;TAB(13)DFGV%(J%,ID%,JD%);TAB(23)DFGT(J%,ID%,JD%
)
11740              NEXT J%
11760              INPUT "Any changes (Y)es or (N)o ";Y$
11780              IF Y$<>"Y" THEN RETURN
11800              PRINT "Enter changes in the format: piece#, value and start
time"
11820              PRINT "Do and enter piece # of 0"
11840              INPUT J%,K%,Y
11860              IF J%=0 GOTO 11640
11880              IF J%>KXMAX%(ID%,JD%) GOTO 11740
```

```
11900             DFGV%(J%,ID%,JD%)=X%: DFGT(J%,ID%,JD%)=Y
11920             GOTO 11840
11940             PRINT "Piece # must not exceed "MAXP%
11960             GOTO 11840
11980   REM
11990   REM
11991   REM ********** WRITE DFGS **********
11992   REM
12000   REM
12020   REM      This subroutine is used to save DFGs that have been entered
12040   REM
12060             IF DRIVE$="B" THEN FILE$="B:"+DFGXX$ ELSE FILE$=DFGXX$
12080             IF DRIVE$="b" THEN FILE$="B:"+DFGXX$
12100             OPEN FILE$ AS #1 LEN=6
12120             FIELD #1, 2 AS V$, 4 AS T$
12140             J4%=0
12160             FOR I2%= 1 TO MU%
12180             FOR J2%= 1 TO NS%
12200             J5%=J4% + 1
12220             LSET V$=MKI$(KXMAX%(I2%,J2%))
12240             PUT #1,J5%
12260             J3% = J4% + 2
12280             J4%=J4% + KXMAX%(I2%,J2%) +1
12300             KD%=1
12320             FOR JD%=J3% TO J4%
12340             LSET V$=MKI$(DFGV%(KD%,I2%,J2%))
12360             LSET T$=MKS$(DFGT(KD%,I2%,J2%))
12380             PUT #1,JD%
12400             KD%=KD%+1
12420             NEXT JD%
12440             NEXT J2%
12460             NEXT I2%
12480             LSET V$ = MKI$(KXMAX%(1,1))
12500             F2 = MU% + NS%/100
12520             LSET T$ = MKS$(F2)
12540             PUT #1,1
12560             CLOSE #1
12580             RETURN
12600   REM
12610   REM
12611   REM ********** READ DFGS **********
12612   REM
12620   REM
12640   REM      ****SUBROUTINE READDFG****
12660   REM      This subroutine is used to read back DFGs that have been previ-
12680   REM      ously entered and saved
12700   REM
12720             IF DRIVE$="B" THEN FILE$="B:"+DFGXX$ ELSE FILE$=DFGXX$
12740             IF DRIVE$="b" THEN FILE$="B:"+DFGXX$
12760             OPEN FILE$ AS #1 LEN=6
12780             FIELD #1, 2 AS V$, 4 AS T$
12800             GET #1,1
12820             F2 = CVS(T$)
12840             SAVEDMU% = F2
```

```
12860             SAVEDNS% = (F2-SAVEDMU%)*100
12880             IF SAVEDMU% <> MU% GOTO 13300
12900             IF SAVEDNS% <> NS% GOTO 13200
12920             J4%=0
12940             FOR I2%=1 TO MU%
12960             FOR J2%= 1 TO NS%
12980             J5% = J4% + 1
13000             GET #1,J5%
13020             KXMAX%(I2%,J2%)=CVI(V$)
13040             J3%=J5%+1
13060             J4% = J4% + KXMAX%(I2%,J2%) + 1
13080             KD% = 1
13100             FOR JD% = J3% TO J4%
13120             GET #1,JD%
13140             DFGV%(KD%,I2%,J2%)=CVI(V$)
13160             DFGT(KD%,I2%,J2%)=CVS(T$)
13180             KD% = KD% + 1
13200             NEXT JD%
13220             NEXT J2%
13240             NEXT I2%
13260             CLOSE #1
13280             RETURN
13300    REM     Dimensions of DFG specified are different from those of
13320    REM     current SAM.
13340             IF SAVEDMU% < MU% GOTO 13460
13360             IF SAVEDNS% < NS% GOTO 13460
13380             PRINT "Dimensions of DFG saved in file exceed those of"
13400             PRINT "current SAM. Simulation will use corresponding portion"
13420             PRINT "of DFG."
13440             GOTO 12920                              'continue reading DFG
13460             PRINT "Dimensions of DFG saved in file are less than those"
13480             PRINT "of current SAM. New DFG may have to be entered."
13500             Q = 1                                   'set error "bit"
13520             CLOSE #1
13540             RETURN
13560    REM
13570   REM
13571   REM ********** ECHO DFGS **********
13572   REM
13580    REM
13600    REM     ****SUBROUTINE ECHODFG****
13620    REM     This subroutine is used for echoing a DFG read from a file.
13640    REM
13660             PRINT: PRINT
13680             PRINT "DFG("I1%","J1%")"
13700             PRINT "Piece #"TAB(10)"Value"TAB(20)"Start Time"
13720             PRINT "--------"TAB(10)"-----"TAB(20)"----------"
13740             FOR J% = 1 TO KXMAX%(I1%,J1%)
13760             PRINT TAB(3)J%;TAB(13)DFGV%(J%,I1%,J1%);TAB(27)DFGT(J%,I1%,J1%)
13780             NEXT J%
13800             INPUT "Any changes (Y)es or any key";X$
13820             IF X$="Y" GOTO 13860
13840             IF X$<>"y" THEN RETURN
```

```
13860             PRINT "Enter changes in the format: piece #, value and start
time"
13880             PRINT "To end enter piece # of 0"
13900             INPUT J%,X%,Y
13920             IF J%=0 GOTO 13680
13940             IF J%>KXMAX%(I1%,J1%) GOTO 14000
13960             DFGV%(J%,I1%,J1%) = X% : DFGT(J%,I1%,J1%) = Y
13980             GOTO 13900
14000             PRINT "Piece # must not exceed "KXMAX%(ID%,JD%)
14020             GOTO 13900
14040   REM
65529 ' THIS IS THE END OF THE MODULE
```

```
 9000    REM                                   LGFOVR.BAS MODULE
 9001    REM ************************************************************
 9002    REM
10000 CLS : LOCATE 2,33 : PRINT "L F   M E N U"
10020 LOCATE 11,25 : PRINT "1 -- Load LF from file"
10040 LOCATE 13,25 : PRINT "2 -- Enter LF from keyboard"
10060 LOCATE 15,25 : PRINT "3 -- Modify LF in memory"
10080 LOCATE 17,25 : PRINT "4 -- Exit to main menu"
10100 LOCATE 23,20 : PRINT SPC(40)
10120 LOCATE 23,20 : INPUT "Enter selection by number ";SUBSEL%
10140 ON SUBSEL% GOTO 10200,10280,10280,5000
10160 SOUND 200,2 " goto 9100
10200 CLS : INPUT "File name from which LF is to be read ";LFXX$
10220 GOSUB 11760
10240 GOTO 10000
10260    REM
10280    REM      Enter Logic Function generator (LF)
10300    CLS
10320             PRINT
10340             GOSUB 10600
10350   REM
10351   REM ********** SAVE LF **********
10352   REM
10360    REM
10380    REM     Prompt to find out if the Logic Function generator just entered
10400    REM     is to be saved.
10420    REM
10440             PRINT
10460             INPUT "Is LF to be saved? (Y)es or No ";Z$
10480             IF Z$="y" GOTO 10520
10500             IF Z$<>"Y" GOTO 5000
10520             PRINT
10540             INPUT "Enter file name under which LF is to be saved. ";LFXX$
10560             GOSUB 11300
10580 GOTO 10000
10590   REM
10591   REM ********** ENTER LF **********
10592   REM
10600    REM      This subroutine is used for entering the array LF (Logic Funct-
10620    REM      ion generator). The array has dimension MU*NS where MU is the
10640    REM      number of input levels and NS is the number of states, as decl-
10660    REM      ared at the beginning of the program. Note that even if the
10680    REM      number of distinct levels in the input signal is less than MU
10700    REM      and/or the number of distinct states appearing in the DFGs is
10720    REM      less than NS, MU*NS entries must be provided for LF.
10725 IF SUBSEL%=3 GOTO 10740
10730 INPUT "Enter number of output levels ";YMAX% : PRINT
10740             FOR J%=1 TO MU%
10750 ON SUBSEL% GOTO 10000,10760, 10860, 5000
10760             PRINT "Enter"NS%" values for output corresponding to input
level ";J%;" and"
10780             PRINT "each of the";NS%;"states."
10800             FOR K%=1 TO NS%
10820             INPUT LF%(J%,K%)
```

```
10840              NEXT K%
10860    REM       Echo the readout map just entered
10880              PRINT "Readout map just entered is as follows:"
10900              PRINT "Input"TAB(10)"State"TAB(20)"Output"
10920              PRINT "-----"TAB(10)"-----"TAB(20)"------"
10940              FOR L%=1 TO NS%
10960              PRINT TAB(2)J%;TAB(12)L%;TAB(22)LF%(J%,L%)
10980              NEXT L%
11000    REM       Ask for modifications, if any
11020              INPUT "Changes? (Y)es or (N)o ";X$
11040              IF X$<>"Y" GOTO 11240
11060              PRINT "Enter changes in the format, state and output."
11080              PRINT "To end, enter state value of 0"
11100              INPUT I%,Y%
11120              IF I%<=0 GOTO 10880
11140              IF I%>NS% GOTO 11200
11160              LF%(J%,I%)=Y%
11180              GOTO 11100
11200              PRINT "State value must not exceed"NS%
11220              GOTO 11100
11240              NEXT J%
11260              RETURN
11280    REM
11290   REM
11291   REM ********** WRITE LF **********
11292   REM
11300    REM       This subroutine is used to save a Logic Function generator file
11320    REM
11340              IF DRIVE$="B" THEN FILE$="B:"+LFXX$ ELSE FILE$=LFXX$
11360              IF DRIVE$="b" THEN FILE$="B:"+LFXX$
11380              OPEN FILE$ AS #1 LEN=2
11400              FIELD #1, 2 AS V$
11420              LSET V$ = MKI$(NS%)
11440              PUT #1,1
11460              LSET V$ = MKI$(MU%)
11480              PUT #1,2
11490              LSET V$ = MKI$(YMAX%)
11495              PUT #1,3
11500              FOR I2% = 1 TO MU%
11520              J3% = (I2%-1)*NS% + 4
11540              J4% = I2%*NS% + 3
11560              KD% = 1
11580              FOR J2% = J3% TO J4%
11600              LSET V$ = MKI$(LF%(I2%,KD%))
11620              PUT #1,J2%
11640              KD% = KD% + 1
11660              NEXT J2%
11680              NEXT I2%
11700              CLOSE #1
11720              RETURN
11740   REM
11750   REM
11751   REM ********** READ LF **********
11752   REM
```

```
11760   REM      This subroutine is used to read a Logic Funtion generator which
11780   REM      has been saved in a file.
11800   REM
11820             IF DRIVE$="B" THEN FILE$="B:"+LFXX$ ELSE FILE$=LFXX$
11840             IF DRIVE$="b" THEN FILE$="B:"+LFXX$
11860             OPEN FILE$ AS #1 LEN=2
11880             FIELD #1,2 AS V$
11900             GET #1,1
11920             SAVEDNS% = CVI(V$)
11940             GET #1,2
11960             SAVEDMU% = CVI(V$)
11980             IF SAVEDNS%<>NS% GOTO 12280
12000             IF SAVEDMU%<>MU% GOTO 12280
12010             GET #1,3 : YMAX%=CVI(V$)
12020             FOR I2% = 1 TO MU%
12040             J3% = (I2%-1)*NS% + 4
12060             J4% = I2%*NS% + 3
12080             KD% = 1
12100             FOR J2% = J3% TO J4%
12120             GET #1,J2%
12140             LF%(I2%,KD%) = CVI(V$)
12160             KD% = KD% + 1
12180             NEXT J2%
12200             NEXT I2%
12220             Q% = 0
12240             CLOSE #1
12260             RETURN
12280             IF SAVEDMU%<MU% GOTO 12380
12300             IF SAVEDNS%<NS% GOTO 12380
12320             PRINT "Dimensions of saved LF exceed those of current SAM."
12340             PRINT "Simulation will use corresponding portion of saved LF."
12360             GOTO 12020
12380             PRINT "Dimension of saved LF is less than that of current SAM."
12400             PRINT "New LF may have to be entered."
12420             PRINT "# states in LF is"SAVEDNS%
12440             PRINT "# input levels in LF is"SAVEDMU%
12460             Q% = 1
12480             CLOSE #1
12500             RETURN
65529 ' this is the end of the program
```

```
9000    REM                          RSMOVR.BAS MODULE
9001    REM ************************************************************
9002    REM
10000    REM      The simulation is event driven, the events being changes
10020    REM      in the input, but NOT the state.  The simulation will
10040    REM      continue to run until either no more such changes occur
10060    REM      (as indicated by the particular DFG signal and input signal
10080    REM      involved) or until a user-specified maximum number of events
10100    REM      have occurred.  We prompt the user for this maximum number
10120    REM      and store it in variable EVENTS%
10140    REM
10160    REM      It is thus true that EVENTS% <= MAXY%.
10170   REM
10171   REM ********** ENTER MAXIMUM EVENTS **********
10172   REM
10180    REM
10200             CLS : PRINT
10220             PRINT " Maximum # of system events in simulation (<="MAXY%")"
10240             PRINT
10260             INPUT "no. of events = ", EVENTS%
10280             IF EVENTS% < 1 GOTO 10320
10300             IF EVENTS% <= MAXY% GOTO 10360
10320             PRINT "Number of events is out of range. Try again."
10340             GOTO 10220
10360    REM
10370   REM
10371   REM ********** ENTER PROCESSING TIME **********
10372   REM
10380    REM      For realizable (physical) SAMs there is a maximum repetition
10400    REM      rate for events which can be processed by the SAM. This arises
10420    REM      from the finite (nonzero) time it takes to process an event.
10440    REM      We call this time TPROC and prompt for it. The default value
10460    REM      is 0, meaning that the SAM is not realizable in practice.
10480    REM
10500             PRINT : PRINT
10520             PRINT "A realizable (physical) SAM takes a finite"
10540             PRINT "length of time to process an event. Enter this time;"
10560             PRINT "the default value is 0, - assumed if CR is entered."
10580             INPUT "Processing time? ",TPROC
10600    REM
10620    REM    Run simulation
10640    REM
10660             PRINT
10680             PRINT "Ready to run simulation. Enter initial state."
10700             PRINT "which must be an integer between 1 and "NS%
10720             INPUT X0%
10740             IF X0% < 1 GOTO 10780
10760             IF X0% <= NS% GOTO 10820
10780             PRINT "Initial state is out of range. Try again."
10800             GOTO 10680
10820    REM    Initial state is in range, run simulation
10860             GOSUB 11230                    'call SUBROUTINE RUNSIM
10880    REM      Compact output array
10900             IND% = INDEX%
```

```
10920              PRINT "IND% = ";IND%
10960     REM     Print output and other arrays
10980              PRINT "Index"TAB(8)"Output value"TAB(24)"Start
time"TAB(38)"Input"TAB(47)"State"
11000              PRINT "-----"TAB(8)"------------"TAB(24)"----------"TAB(38)"---
-
"TAB(47)"------"
11020              FOR J% = 1 TO IND%
11040              PRINT TAB(3)
J%;TAB(13)YV%(J%);TAB(27)YT(J%);TAB(39)URES%(J%);TAB(47)XRES%(J%)
11060              NEXT J%
11080              PRINT
11100              PRINT "Results from this run may be saved in a file on disk."
11120              PRINT "If results are to be saved, enter a file name in the"
11140              PRINT "form 'RESxx', where xx are numeric characters."
11160              PRINT "If results are NOT to be saved enter CR at the prompt."
11180              INPUT "File name for saving results. ",RESXX$
11200              IF RESXX$="" GOTO 11240
11220              GOSUB 13300
11240              GOTO 5000
11260     REM
11270    REM
11271    REM ********** RUN SIMULATION **********
11272    REM
11280     REM      ****SUBROUTINE RUNSIM****
11300     REM      This subroutine performs the actual simulation on the model
11320     REM      entered in the foregoing part of the program. The simulation
11340     REM      consists of determining the output at every instant as well as
11360     REM      the corresponding state and input. Since the model assumes that
11380     REM      changes in the output can only occur when there is a change
11400     REM      either in the input or in the state, the simulation is event
11420     REM      driven.
11440     REM
11460     REM      Set the time corresponding to an infinite delay.
11480     REM      Also set EPSI, a small positive quantity, for use in
11500     REM      determining when TU and TX are about equal.
11520     REM      CD% is a code used to determine whether an input event
11540     REM      occurred during processing, throughout the simulation.
11560               TINF = 32000 : EPSI = .000001 : CD% = 0
11580     REM      Initialize indices
11600               INDEX%=1: IX%=1: IU%=1
11620     REM      Initialize input and DFG
11640               UI%=UV%(IU%)
11660               XL%=XO%
11680     REM      Initialize state
11700               XJ%=DFGV%(1,UI%,XL%)
11720     REM      First output piece
11740               YT(1)=0 :YV%(1)=LF%(UI%,XJ%)
11760               URES%(1)=UI%: XRES%(1)=XJ%
11780     REM      Start loop
11800               INDEX% =2
11820     REM      Get time to next transition on input and state
11840               IF NUMPU%<2 THEN TU=TINF ELSE TU=UT(2)
11860               IF DXMAX%(UI%,XL%)<2 THEN TX=TINF ELSE TX=DFGT(2,UI%,XL%)
11880               IX%=2
```

```
11900     REM      Test for occurence of an event during processing time.
11920     REM      Note that we test only for occurence of an input change
11940     REM      since a state change is not considered a maskable event.
11960               Q% = 0                              'clear test bit.
11980               IF TU >= TPROC GOTO 12280
12000               Q% = 1 : CD% = 1                    'event during processing.set Q%
12020     REM      Current input does not persist throughout processing time.
12040     REM      Get next input piece.
12060               IU% = IU% + 1
12080               IF IU% >= NUMPU% THEN TU = TINF
ELSE TU = TU + UT(IU%+1) - UT(IU%)
12100               IF IU% > NUMPU% THEN IU% = NUMPU%
12120     REM      Check to see if duration of new piece(s) plus what is left of
12140     REM      current piece is up to TPROC.
12160               IF TU < TPROC GOTO 12040
12180     REM      TU is now large enough. Decrement input piece pointer. This
12200     REM      will be incremented in the part of the program which handles
12220     REM      the case where the next event is an input change.
12240               IU% = IU% - 1
12260               GOTO 12760
12280     REM      Check for smaller of TU and TX
12300               IF ABS(TU-TX) < EPSI GOTO 13220
12320               IF TU < TX GOTO 12760
12340     REM      Next event is a change in the state. Get new state.
12360               XJ%=DFGV%(IX%,UI%,XL%)
12380     REM      Get output value and start time.
12400               YV%(INDEX%)=LF%(UI%,XJ%)
12420               YT(INDEX%)=TX+YT(INDEX%-1)
12440               IF Q%=1 THEN YT(INDEX%)=YT(INDEX%-1)+TPROC
12460               URES%(INDEX%)=UI%: XRES%(INDEX%)=XJ%
12480     REM      Increment output piece pointer.
12500               INDEX%=INDEX%+1 :IF INDEX%<=EVENTS% GOTO 12540
12520               INDEX% = INDEX% - 1: RETURN
12540     REM      Adjust TU.
12560               IF TU<>TINF THEN TU=TU-TX
12580     REM      Read new TX.
12600               IF Q% = 0 GOTO 12660
12620               IF TX <> TINF THEN TX = TX - TPROC
12640               GOTO 11900
12660               IX%=IX%+1
12680               IF IX%<=KXMAX%(UI%,XL%) GOTO 12720
12700               IX% = IX% - 1: TX = TINF: GOTO 11900
12720               TX=DFGT(IX%,UI%,XL%)-DFGT(IX%-1,UI%,XL%)
12740               GOTO 11900
12760     REM      Next event is an input change. Get new input and DFG.
12780               IU% = IU% + 1
12800               UI%=UV%(IU%)
12820               XL%=XJ%
12840     REM      Get new state
12860               IX%=1: XJ%=DFGV%(1,UI%,XL%)
12880     REM      Get output
12900               YV%(INDEX%)=LF%(UI%,XJ%)
12920               YT(INDEX%)=TU+YT(INDEX%-1)
12940               IF Q%=1 THEN YT(INDEX%)=YT(INDEX%-1)+TPROC
```

```
12960             URES%(INDEX%)=UI%: XRES%(INDEX%)=XJ%
12980             INDEX%=INDEX%+1
13000             IF INDEX%<=EVENTS% GOTO 13040
13020             INDEX% = INDEX% - 1: RETURN
13040   REM      Increment state pointer.
13060             IX%=IX%+1
13080             IF IX%<=KXMAX%(UI%,XL%) GOTO 13140
13100             TX=TINF: IX%=IX%-1
13120             GOTO 13160
13140             TX=DFGT(2,UI%,XL%)
13160   REM      Get new TU
13180             IF (IU%+1)>NUMPU% THEN TU=TINF ELSE TU=UT(IU%+1)-YT(INDEX%-1)
13200             GOTO 11900
13220   REM      TU=TX.
13240             IF TU<>TINF GOTO 12760
13260             INDEX% = INDEX% -1: RETURN
13280   REM
13300   REM
13301   REM ********** SAVE SIMULATION RESULTS **********
13302   REM
13320   REM      This subroutine is used for saving the results of a run.
13340   REM
13360             IF DRIVE$="B" THEN FILE$="B:"+RESXX$ ELSE FILE$=RESXX$
13380             IF DRIVE$="b" THEN FILE$="B:"+RESXX$
13400             OPEN FILE$ AS #1 LEN=10
13420             FIELD #1, 2 AS Y$, 2 AS U$, 2 AS X$, 4 AS T$
13440   REM      Save specifics of the run
13460             LSET U$ = RIGHT$(INPUTXX$,2)
13480             LSET X$ = RIGHT$(DFGXX$,2)
13500             LSET Y$ = RIGHT$(LFXX$,2)
13520             LSET T$ = MKI$(IND%)
13540             PRINT Y$,U$,X$,T$
13560             PUT #1,1
13580             LSET Y$ = MKI$(YMAX%)
13600             LSET U$ = MKI$(MU%)
13620             LSET X$ = MKI$(NS%)
13640             LSET T$ = MKI$(XO%)
13660             PUT #1,2
13680             LSET Y$ = MKI$(CD%)
13700             LSET T$ = MKS$(TPROC)
13720             PUT #1,3
13740             FOR J% = 4 TO IND%+3
13760             LSET Y$ = MKI$(YV%(J%-3))
13780             LSET U$ = MKI$(URES%(J%-3))
13800             LSET X$ = MKI$(XRES%(J%-3))
13820             LSET T$ = MKS$(YT(J%-3))
13840             PUT #1,J%
13860             NEXT J%
13880             CLOSE #1
13900             RETURN
65529 REM
```

```
9000     REM                          PLTOVR.BAS MODULE
9001     REM ************************************************************************
9002     REM
10000 CLS : LOCATE 2,31 : PRINT "P L O T   M E N U"
10020 LOCATE 9,25 : PRINT "1 -- Plot input function"
10040 LOCATE 11,25 : PRINT "2 -- Plot state function"
10060 LOCATE 13,25 : PRINT "3 -- Plot output function"
10080 LOCATE 15,25 : PRINT "4 -- Plot DFGs"
10100 LOCATE 17,25 : PRINT "5 -- Load results from file to plot"
10120 LOCATE 19,25 : PRINT "6 -- Exit to main menu"
10140 LOCATE 23,20 : PRINT SPC(40)
10160 LOCATE 23,20 : INPUT "Enter selection by number ": SUBSEL%
10180 ON SUBSEL% GOTO 10440, 10600, 10300, 10780, 10220, 5000
10200 SOUND 200,2 : GOTO 10140
10220 CLS : INPUT "Enter results file name ";RESXX$
10240 GOSUB 13680 : GOTO 10000
10260    REM
10280             CLS                                'clear screen
10290    REM
10291    REM ********** PLOT OUTPUT **********
10292    REM
10300     REM      Output is to be saved.
10320              FOR L% = 1 TO IND%
10340              FPLT%(L%) = YV%(L%)
10360              NEXT L%
10380              NAM% = 1
10400              LEVELS% = YMAX%
10420              GOTO 10840
10430    REM
10431    REM ********** PLOT INPUT **********
10432    REM
10440     REM      Input is to be saved.
10460              FOR L% = 1 TO IND%
10480              FPLT%(L%) = URES%(L%)
10500              NEXT L%
10520              NAM% = 2
10540              LEVELS% = MU%
10560              CODE% = CD%
10580              GOTO 10840
10590    REM
10591    REM ********** PLOT STATE **********
10592    REM
10600     REM      State is to be saved.
10620              FOR L% = 1 TO IND%
10640              FPLT%(L%) = XRES%(L%)
10660              NEXT L%
10680              NAM% = 3
10700              LEVELS% = NS%
10760              GOTO 10840                         'skip to [illegible]
10770    REM
10771    REM ********** PLOT DFGS **********
10772    REM
10780     REM      DFG is to be plotted.
10800              GOSUB 14300
```

```
10820              SCREEN 0 : GOTO 10000          'set screen to text mode.
10840    REM       Save time array and specifics of the run.
10860              SP1$ = INPUTXX$ + "."
10880              SP2$ = DFGXX$ + "."
10900              SP3$ = LFXX$ + ".Init. state"
10920              SPECS$ = SP1$ + SP2$ + SP3$
10940              FOR L% = 1 TO IND%
10960              TIMEPLT(L%) = YT(L%)
10980              NEXT L%
11050    REM
11051    REM ********** PLOT REQUESTED FUNCTION **********
11052    REM
11060     REM      Functions saved are to be plotted now.
11080              PRINT
11140     REM     For each function to be plotted, fill in level array and time
11160     REM     array.
11260              FOR KL% = 1 TO IND%
11280              VALL%(KL%) = FPLT%(KL%)
11300             TIM(KL%) = TIMEPLT(KL%)
11320             NEXT KL%
11340             NP% = IND%
11360             N% = LEVELS%
11380             CAPTION% = NAM%
11400             STAR% = CODE%
11420             TP = TPROC
11440             FLS$ = SPECS$
11460             GOSUB 11620
11520             SCREEN 0                          'screen to text mode
11580             GOTO 10000
11600             END
11620     REM
11630    REM
11631    REM ********** PLOTTER FUNCTION **********
11632    REM
11640     REM              ****SUBROUTINE PLOTTER****
11660     REM
11680     REM      THIS ROUTINE PLOTS OUT A PIECEWISE CONSTANT FUNCTION OF FINITE
11700     REM      RANGE. THE FUNCTION IS ASSUMED TO HAVE AT MOST [illegible] LEVELS AND
11720     REM      AT MOST 100 PIECES.  IT USES STANDARD HIGH RESOLUTION GRAPHICS
11740     REM      ON THE AT&T.
11760     REM
11780     REM      IN HIGH RESOLUTION GRAPHICS THE SCREEN IS 640 BY 200 WITH
11800     REM      POINT (0,0) IN THE UPPER LEFT HAND CORNER AND POINT (639,199)
11820     REM      IN THE LOWER RIGHT. ASSUME THAT THE VALUES (LEVELS) OF
11840     REM      THE FUNCTION RUN FROM 1 TO N, N <= 8, AND THAT LEVEL 1 GOES TO
11860     REM      WILL APPEAR ALONG PIXEL COORDINATES
11880     REM      (...,175-J*20). TIME VALUES ARE HANDLED AS FOLLOWS:
11900     REM      EACH TIME FUNCTION TO BE PLOTTED WILL BE SPECIFIED BY [illegible]

11920     REM      --SEE BELOW--. THE TIME ARRAY HAS AS ITS LAST ELEMENT [illegible]
11940     REM      TIME FOR THE LAST PIECE OF THE FUNCTION.  [illegible]
11960     REM      FOR THE PURPOSES OF THE CURRENT DISCUSSION, [illegible]
11980     REM      TLAST TO CO-INCIDE WITH HORIZONTAL [illegible]
12000     REM      TIME 0 (THE START TIME OF THE FIRST PIECE OF [illegible]
12020     REM      TO CORRESPOND TO HORIZONTAL PIXEL VALUE OF [illegible]
```

```
12040    REM     TO <= TLAST APPEARS AT HORIZONTAL PIXEL VALUE
12060    REM     OF 20 + (TO/TLAST)*500.
12080    REM
12100    REM     IF TLAST=0 THEN THE TIME FUNCTION HAS BUT ONE PISCE.  THIS
12120    REM     CASE REQUIRES NO SCALING OF THE HORIZONTAL AXIS, OBVIOUSLY,
12140    REM     AND IS SPECIALLY HANDLED BELOW.
12160    REM
12180    REM
12200    REM     CHANGES IN THE TIME FUNCTION ARE HANDLED AS FOLLOWS:
12220    REM     IF THE FUNCTION CHANGES FROM VALUE K TO VALUE J AT TIME T
12240    REM     THEN WE WILL SEE THREE LINES -- A HORIZONTAL ONE RUNNING AT
12260    REM     VERTICAL PIXEL VALUE 175-K*20 UP TO POINT
12280    REM     (20+(T/TLAST)*500,175-K*20), A VERTICAL LINE FROM THERE
12300    REM     TO (20+(T/TLAST)*500,175-J*20), AND ANOTHER
12320    REM     HORIZONTAL LINE RUNNING TO THE RIGHT FROM THAT POINT.
12340    REM
12360    REM     THE PROGRAM DRAWS A SET OF LABELLED AXES AS WELL.  THIS
12380    REM     PROCEDURE WILL BE SELF-EXPLANATORY IN THE CODE
12400    REM
12420             SCREEN 0      'SCREEN TO TEXT MODE
12440    REM
12460    REM     SET UP FOR HIGH RESOLUTION GRAPHICS
12480    REM
12500             CLS: SCREEN 2 : KEY OFF
12520    REM
12540    REM     DETERMINE TLAST
12560    REM
12580             TLAST = TIM(NP%)
12600             IF CAPTION%=1 THEN CAP$ = "OUTPUT FUNCTION"
12620             IF CAPTION%=2 THEN CAP$ = "INPUT FUNCTION"
12640             IF CAPTION% <> 2 GOTO 12720
12660             IF STAR% = 1 THEN CAP$ = CAP$ + "*"
12680             LOCATE 1,25 : PRINT CAP$
12700             GOTO 12840
12720             IF CAPTION%=3 THEN CAP$ = "STATE FUNCTION"
12740             IF CAPTION% <> 4 GOTO 12820
12760             CAP$ = DFGXX$
12780             LOCATE 1,25 :PRINT CAP$; TAB(36)"Element ";I%",";J%.
12800             GOTO 12840
12820             LOCATE 1,8: PRINT CAP$;TAB(29) PLS$;" ..."  pr    time: "T
12840    REM
12860    REM     DRAW AND LABEL Y-AXIS AND HASH MARKS
12880    REM
12900             LINE (20,175) - (20,15)
12920             IF N%<9 THEN D%=N% ELSE D%=9
12940             W% = 160/D%
12960             FOR J% = 1 TO N%
12980             LOCATE (175-J%*W%)/8,1 : PRINT USING "##";J%;
13000             LINE (18,175-J%*W%)-(22,175-J%*W%)
13020             NEXT J%
13040    REM
13060    REM     DRAW AND LABEL X-AXIS.  SINCE HORIZONTAL PIXEL VALUES
13080    REM     OF 20 AND 520 CORRESPOND TO TIME VALUES OF 0 AND TLAST
13100    REM     RESPECTIVELY WE WILL PLOT VALUES TLAST*(0,1*I)...
```

```
13120     REM      AT PIXEL LOCATIONS 70,120,... WITH VALUE J*TLAST/10 AT
13140     REM      LOCATION 470.  THE TLAST CASE IS HANDLED SPECIALLY.
13160     REM
13180               LINE (20,175) - (639,175)
13200               IF TLAST = 0 THEN GOTO 13320
13220               FOR J% = 0 TO 10
13240               LOCATE 25,(20+J%*50)/8 : PRINT J%*TLAST/10! ;
13260               LINE (20+J%*50,173) - (20+J%*50,177)
13280               NEXT
13300               GOTO 13340
13320               LOCATE 25,20 : PRINT 0          ' TLAST = 0 CASE
13340     REM
13360     REM      NOW WE DRAW THE WAVEFORM.  A SINGLE LOOP WILL DRAW
13380     REM      ALL BUT THE LAST PIECE, WHICH IS DRAWN SEPARATELY.
13400     REM      THE NP%=1 CASE IS HANDLED IN A SPECIAL WAY.
13420     REM
13440               LINE (20,175-VALL%(1)*W%)-(20,175-VALL%(1)*W%)
13460               IF NP%=1 THEN GOTO 13560
13480               FOR J% = 1 TO NP%-1
13500               LINE - (20+TIM(J%+1)/TLAST*500,175-VALL%(J%)*W%)
13520               LINE - (20+TIM(J%+1)/TLAST*500,175-VALL%(J%+1)*W%)
13540               NEXT J%
13560               LINE - (639,175-VALL%(NP%)*W%)
13580     REM
13600     REM      PUT IN 'ESC' MECHANISM FOR PRINTING AND LOOP BACK
13620     REM
13640               IF INKEY$=CHR$(27) THEN RETURN ELSE GOTO 13640
13660               END
13680    REM
13690    REM
13691    REM ********** LOAD RESULTS FILE **********
13692    REM
13700    REM      This subroutine is used for reading the results of a run
13720    REM      which have been saved.
13740    REM
13760               IF DRIVE$="B" THEN FILE$="B:"+RESXX$ ELSE FILE$=RESXX$
13780               IF DRIVE$="b" THEN FILE$="B:"+RESXX$
13800               OPEN FILE$ AS #1 LEN=10
13820               FIELD #1, 2 AS Y$, 2 AS U$, 2 AS X$, 4 AS T$
13840    REM      Read specifics of the run
13860               GET #1,1
13880               INPUTXX$="INPUT"+U$
13900               DFGXX$="DFG"+X$
13920               LFXX$="LF"+Y$
13940               IND% = CVI(T$)
13960               GET #1,2
13980               YMAX% = CVI(Y$)
14000               MU% = CVI(U$)
14020               NS% = CVI(X$)
14040               X0% = CVI(T$)
14060               GET #1,3
14080               CD% = CVI(Y$)
14100               TPROC = CVS(T$)
14120               FOR J% = 4 TO IND%+3
```

```
14140             GET #1,J%
14160             YV%(J%-3) = CVI(Y$)
14180             URES%(J%-3) = CVI(U$)
14200             XRES%(J%-3) = CVI(X$)
14220             YT(J%-3) = CVS(T$)
14240             NEXT J%
14260             CLOSE #1
14280             RETURN
14290    REM
14291    REM ********** ADJUST DFG TIME SCALE **********
14292    REM
14300     REM     This subroutine is used to interface a DFG with the
14320     REM     plotting subroutine. The time scale of the plot must be
14340     REM     adjusted, (usually, this will mean expanded', to coincide
14360     REM     with that of the output function which resulted from the
14380     REM     use of this DFG.
14400              FOR JP% = 1 TO SAVEDMU%
14420              FOR KP% = 1 TO SAVEDNS%
14440              T% = KXMAX%(JP%,KP%)
14460              IF DFGT(T%,JP%,KP%) > YT(IND%) GOTO 14540
14480              DFGV%(T%+1,JP%,KP%) = DFGV%(T%,JP%,KP%)
14500              DFGT(T%+1,JP%,KP%) = YT(IND%)
14520              GOTO 14560
14540              T% = T%-1
14560              FOR I% = 1 TO T%+1
14580              VALL%(I%) = DFGV%(I%,JP%,KP%)
14600              TIM(I%) = DFGT(I%,JP%,KP%)
14620              NEXT I%
14640              NP% = T% + 1
14660              N% = SAVEDNS%
14680              CAPTION% = 4
14700              GOSUB 11620
14720              NEXT KP%
14740              NEXT JP%
14760              RETURN
15529 'this is the end of this module
```

6.2 Additional Publications

Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
Vol 31, No. 1, August 1985
Printed in Belgium

Behavior of a Class of Nonlinear Discrete-Time Systems

MARTIN E. KALISKI*

Department of Electrical and Computer Engineering, Northeastern University, Boston, Massachusetts 02115

AND

QUENTIN L. KLEIN

25 Milo Street, West Newton, Massachusetts 02165

Received August 10, 1983; accepted March 15, 1985

A novel collection of nonlinear discrete-time systems is analyzed and characterized up to isomorphism. The systems in question are autonomous and deterministic and have the half-open unit interval [0, 1) as state space. The analysis establishes isomorphisms between these systems and the members of an especially simple, easy-to-understand, normal-form subclass of "prototype" systems. The prototypes consist exclusively of piecewise-linear systems with parallel pieces. The theory hinges upon a generally unfamiliar but remarkable real number representation resembling ordinary binary or decimal notation but involving a radix or base which is not an integer.

I. INTRODUCTION

The systems to be considered here are discrete-time systems having as state space the half-open unit interval [0, 1) of the real line, and are autonomous, time-invariant, and deterministic, as well. Thus they are characterized by a difference equation of the form $x_{k+1} = f(x_k)$, $k = 0, 1, 2, \ldots$, where f is a function mapping [0, 1) to [0, 1). We further require f to have the general form shown in Fig. 1 (and described precisely in Section II below).

When started at time zero in an initial state x_0 such systems will naturally proceed through an infinite sequence of states x_0, $x_1 = f(x_0)$, $x_2 = f(x_1)$ and so on (Fig. 2). This sequence constitutes the "orbit" of the system arising from initial state x_0. This paper will investigate how qualititative aspects of the orbit change, first as x_0 is varied through [0, 1) and second, as f itself is changed (but still of the form of Fig. 1).

By focussing attention on a subset of these systems—the "signature-distinct" ones (see Sect. II)—we shall develop an isomorphic relationship between these systems

* This author's work was supported in part by the U.S. Air Force Office of Scientific Research (AFSC), Contract F49620-82C-0080.

0022-0000/85 $3.00

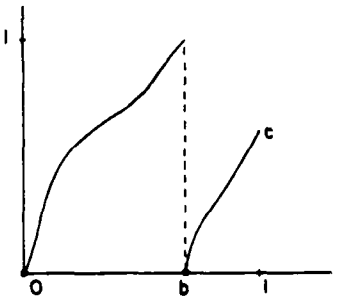


FIG. 1. A typical saw function.

and piecewise-linear systems. This isomorphic relationship will essentially reduce the study of nonlinear signature-distinct systems to the study of piecewise-linear prototypes with parallel pieces—a great simplification, since the latter are much more manageable, both computationally and conceptually.

Three diverse considerations motivate the approach of the present paper. The first is strictly pragmatic: extending prior research by the authors in this area for similar systems having continuous transition functions [3, 4, 5]. Our orbit space here is much more intuitively structured and ordered and the net result is a detailed structure theory for these systems—an achievement which we believe will lead the way to similar results for more extensive classes of systems (see Section VI). The second motivation involves demonstration that the analytic techniques of our earlier papers are not restricted to continuous transition functions and as such are applicable to systems possessing two distinct modes of behavior as in Fig. 1. The third motivation is purely mathematical: an old problem of Ulam [13] asks whether any continuous transformation of $(0, 1)$ is conjugate to a piecewise-linear transformation of same. The results presented here represent additional progress on this basic mathematical question.

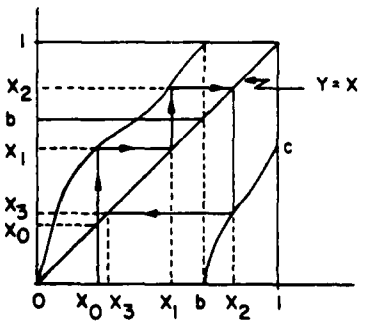


FIG. 2. The orbit arising from a point x_0.

Prior research in this area has employed the tools of ergodic theory [11] or has restricted itself to the study of fixed points of the iterates of functions [9, 6]. Our approach here is deliberately fundamental, concentrating on the isomorphism question and using only the most elementary tools of mathematical analysis. The authors are currently endeavoring to reconcile the more esoteric methods of system analysis with the approach presented here. The interested reader should consult the monograph of Collet and Eckmann [1] for a good overview of current work in this dynamic field, as well as the now "classic" papers of Li and Yorke [7] and May [8].

II. Saw Functions, Signatures, and Fundamental Results

We propose to investigate the behavior of 1-dimensional autonomous discrete-time systems possessing saw functions for their transition maps. These functions have the sawtooth form illustrated in Fig. 1 and are formally defined as follows:

Definition 1. A function $f: [0, 1) \to [0, 1)$ is a saw function iff there exists a point b in $(0, 1)$ such that the following three conditions hold:

(a) f is continuous and strictly monotone increasing on each of the intervals $[0, b)$ and $[b, 1)$,

(b) $f(0) = 0$ and $f(b) = 0$,

(c) the limit of $f(x)$ as x approaches b from the left is 1 and the limit of $f(x)$ as x approaches 1 from the left is some quantity c in $(0, 1]$.

Note that NO assumption of differentiability, linearity, or convexity is placed upon any part of f. The point b will be called the "breakpoint" of f.

Definition 2. Two saw functions f and g are said to be isomorphic iff there exists a (necessarily strictly monotone increasing) homeomorphism $h: [0, 1) \to [0, 1)$ such that $f = hgh^{-1}$, where h^{-1} denotes the inverse of h.

The isomorphism of two saw functions f and g immediately implies that the autonomous discrete-time systems they define are essentially the same as regards iterative, order-theoretic, and topological properties. The following further definitions are fundamental in pursuing this line of thought.

Definition 3. Given an arbitrary function f denote by f^k, $k = 1, 2, 3, \ldots$, the k-fold composition $f^k(x) = f(f(\cdots f(x)\cdots))$. Take $f^0(x) = x$. The function f^k will be called the kth-iterate of f.

Definition 4. Let f be an arbitrary saw function and let x be a point in $[0, 1)$. Define the orbit of x under f, denoted by $\mathrm{orb}_f(x)$, to be the infinite sequence of real numbers $x_0 x_1 \ldots$, where $x_k = f^k(x)$, for $k = 0, 1, \ldots$. Define the signature of x under f,

denoted by $\text{sig}_f(x)$, to be the infinite binary sequence $s_0 s_1 \ldots$, where $s_k = 0$ if x_k is in $[0, b)$ and $s_k = 1$ if x_k is in $[b, 1)$.

Thus $\text{sig}_f(x)$ is a simple 2-level quantization of $\text{orb}_f(x)$ which serves to indicate to which of the two subintervals $[0, b)$ or $[b, 1)$ each x_k belongs. The analysis below will demonstrate that signatures are more useful in understanding the orbital behavior of f than are the orbits themselves; the superabundance of exact quantitative information in orbits obscures the intrinsic, coordinate-system-independent structural information which signatures expose!

The following theorem, although fundamental, is straightforward and its proof is omitted.

THEOREM 1. *Let* $h: [0, 1) \to [0, 1)$ *be a homeomorphism and* f *a saw function with breakpoint* b. *If* g *is the function defined by* $g = hfh^{-1}$ *then* g *is also a saw function, with breakpoint* $h(b)$ *and, for all* x *in* $[0, 1)$, $\text{sig}_f(x) = \text{sig}_g(h(x))$.

DEFINITION 5. For any saw function f define the orbit repertoire of f, denoted as OR_f, as the set $\{\text{orb}_f(x) \mid x \text{ is in } [0, 1)\}$ and define the signature repertoire of f, SR_f, as $\{\text{sig}_f(x) \mid x \text{ is in } [0, 1)\}$.

COROLLARY. *If* $g = hfh^{-1}$, *as in Theorem* 1, *then* $\text{SR}_f = \text{SR}_g$.

Note. If $g = hfh^{-1}$ as in Theorem 1 it is manifest that for corresponding points x and $h(x)$, $\text{orb}_f(x)$ can differ arbitrarily from $\text{orb}_g(h(x))$, whereas the signatures have been shown to be identical. This illustrates our earlier remark about the invariance of signatures under a "change of coordinates."

DEFINITION 6. Let S denote the set of all infinite binary sequences $s_0 s_1 s_2 \ldots$. Consider S to be ordered lexicographically and let "$<$" denote this ordering. That is, $s = s_0 s_1 s_2 \ldots < t = t_0 t_1 t_2 \ldots$ if they are not identical and if, setting k to be the first position in which they differ, $s_k = 0$ and $t_k = 1$. Use $<$ to define intervals (s, t), $[s, t)$, $(s, t]$, and $[s, t]$ in the usual fashion. Consider all possible open intervals (s, t), together with the "semi-infinite" intervals $\{t \mid t < s\}$ and $\{t \mid t > s\}$ as defining a topology TOP on S.

The reader's familiarity with the following mathematical properties of S, $<$, and TOP is assumed:

(a) S is uncountable

(b) $<$ really is an order relation

(c) $<$ is a total order on S

(d) S contains a least element, 000..., and a greatest element, 111...

(e) given any subset T of S, there exists a unique element of S serving as $\sup(T)$ (provided of course that T is non-empty)

(f) similarly $\inf(T)$ always exists, if T is non-empty

(g) the topology TOP is a standard one on totally ordered sets known as the "order topology"

(h) in TOP a sequence s^i of elements of S converges to s in S iff for each k the sequence of kth terms of the s^i becomes eventually constant, with value equal to the kth term of s

(i) S is not homeomorphic to $[0, 1]$ owing to the presence of gaps in the order: there are no elements of S between each pair of elements of the form $s_0 s_1 \ldots s_k\, 0\, 1\, 1\, 1 \ldots$, and $s_0 s_1 \ldots s_k\, 1\, 0\, 0\, 0 \ldots$.

Since signatures are infinite binary sequences themselves we will consider them as elements of S. The significance of the $<$ order thus placed on signatures will be explained by the next theorem; the significance of the topology will become evident in Section III.

DEFINITION 7. For s in S, $s = s_0 s_1 s_2 \ldots$, define for $k \geq 1$ the sequence $\mathrm{lop}^k(s)$ to be the sequence $s_k s_{k+1} \ldots$, obtained by lopping off and throwing away the first k terms of s. Define $\mathrm{lop}(s)$ to be $\mathrm{lop}^1(s)$. Define $\mathrm{lop}^0(s)$ to be s.

The following obvious property will be referred to as the "mapping property" for signatures and will be useful throughout the sequel:

$$\mathrm{sig}_f(f^k(x)) = \mathrm{lop}^k(\mathrm{sig}_f(x)) \qquad \text{for all} \quad k \geq 0$$

DEFINITION 8. Given an arbitrary saw function f with breakpoint b define f_0 as the function obtained by restricting f to $[0, b)$ and define f_1 as the function obtained by restricting f to $[b, 1)$. Note that f_0 and f_1 are both strictly monotone increasing on their respective domains.

THEOREM 2 (Monotonicity of signatures property). *Let a fixed saw function f be given. For arbitrary points x and y in $[0, 1)$, $x < y$, it is always the case that* $\mathrm{sig}_f(x) \leq \mathrm{sig}_f(y)$.

Proof. Recall the above definition of f_0 and f_1. Given an arbitrary point z in $[0, 1)$ with $\mathrm{orb}_f(z) = z_0 z_1 z_2 \ldots$, and $\mathrm{sig}_f(z) = w_0 w_1 w_2 \ldots$, we have the general formula

$$z_k = f^k(z) = f_{w_{k-1}} f_{w_{k-2}} \cdots f_{w_0}(z),$$

since the signature of z tells us which "partial" function f_0 or f_1 to use in calculating z_{i+1} from z_i, for $i = 0, 1, \ldots$.

Now consider the given points x and y. If $\mathrm{sig}_f(x) = \mathrm{sig}_f(y)$ we are done. If the signatures differ set $\mathrm{sig}_f(x) = s_0 s_1 s_2 \ldots$, $\mathrm{sig}_f(y) = t_0 t_1 \ldots$, and let k be the least index for which they differ. If $k \geq 1$ then apply the above formula to conclude that

$$\begin{aligned} f^{(k)}(x) &= f_{s_{k-1}} \cdots f_{s_1} f_{s_0}(x) \\ f^k(y) &= f_{t_{k-1}} \cdots f_{t_1} f_{t_0}(y) \\ &= f_{s_{k-1}} \cdots f_{s_1} f_{s_0}(y) \end{aligned}$$

by the definition of k. The functional composition common to the right-hand sides of these expressions is a strictly monotone increasing function, being the composition of such, so $x < y$ implies $f^k(x) < f^k(y)$. Note that this is true even if $k = 0$. Hence in all cases the statement "s_k not equal to t_k" can be valid only if $s_k < t_k$. Thus $\text{sig}_f(x) < \text{sig}_f(y)$ by the definition of the $<$ order on signatures.

Hence in all cases we have $\text{sig}_f(x) \leqslant \text{sig}_f(y)$. Q.E.D.

The monotonicity of signatures property shows that the order in which a saw function assumes its signatures is pre-determined. This simple regularity will be exploited throughout the remainder of the paper. The remainder of this section discusses a number of corollaries to Theorem 2. Most of the proofs are straightforward and will be omitted for lack of space.

MONOTONICITY COROLLARY 1. *Given a saw function f the following are all equivalent formulations of the Monotonicity of Signatures Principle*:

(a) sig_f: $[0, 1] \to S$ *is a (generally non-strictly) monotone increasing map*

(b) *if* $\text{sig}_f(x) > \text{sig}_f(y)$ *then* $x > y$

(c) *it is impossible to have* $x < y$ *and* $\text{sig}_f(x) > \text{sig}_f(y)$ *simultaneously.*

It is important to understand that one cannot in general strengthen the statement of Theorem 2 to read "$x < y$ implies $\text{sig}_f(x) < \text{sig}_f(y)$." The following discussion will address this latter point.

DEFINITION 9. A saw function f is said to signature-distinct if and only if for all x and y in $[0, 1)$, x not equal to y implies $\text{sig}_f(x)$ not equal to $\text{sig}_f(y)$.

Figure 3 establishes the existence of saw functions that are not signature-distinct: the shaded x-axis interval sits entirely to the left of the breakpoint b and maps into itself under the action of the saw function shown. Hence all points of the shaded interval have signature 000 ....

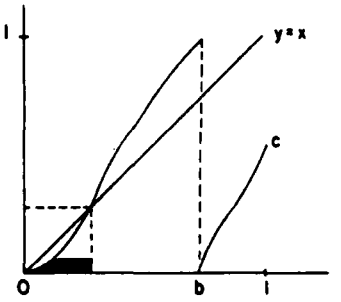


FIG. 3. A saw function which is not signature-distinct.

DEFINITION 10. For any saw function f and any sequence s in S, define $p_f(s)$ to be the set of points x in $[0, 1)$ for which $\text{sig}_f(x) = s$.

MONOTONICITY COROLLARY 2. *For any saw function f and any s in S, $p_f(s)$ is either empty, a single point or an interval.*

In the next corollary we do indeed establish the existence of a wide class of signature-distinct saw functions:

DEFINITION 11. Let f be a given saw function, f_0, f_1 as in Definition 8. Define F_0 on domain $[0, b]$ by setting $F_0(b) = 1$; $F_0(x) = f_0(x)$ for x in $[0, b)$. Similarly define F_1 on domain $[b, 1]$ by defining $F_1(1) = c$, $F_1(x) = f_1(x)$ for x in $[b, 1)$. We say f is piecewise strictly expansive (pse) if and only if F_0 and F_1 are strictly expansive in their actions, i.e., for arbitrary x and y in the domain of F_0, $|F_0(x) - F_0(y)| > |x - y|$ and similarly for F_1.

MONOTONICITY COROLLARY 3. *If a saw function is pse it is signature-distinct.*

Proof by contradiction. Assume that $s = s_0 s_1 s_2 \ldots$ is a signature under f for which $p_f(s)$ is an interval of positive length. Observe that for all $i \geq 0$ by the mapping property for signatures, $F_{s_i}(p_f(\text{lop}^i(s)))$ is a subset of $p_f(\text{lop}^{i+1}(s))$. By the expansive nature of F_0 and F_1 then we conclude that $0 < \text{length } p_f(s) < \text{length } p_f(\text{lop}(s)) < \text{length } p_f(\text{lop}^2(s)) < \ldots$.

Two subcases arise: if all the signatures $\text{lop}^i(s)$ are different from each other, and from s, then all of the intervals in the inequality chain above are disjoint, so that the sum of all of their lengths must be at most 1. Yet the sequence of lengths is increasing—contradiction. So it must be that $\text{lop}^i(s) = \text{lop}^j(s)$ for some i and j. In this situation we would have equality of lengths, contradicting the inequality of lengths noted above.

The only conclusion: $p_f(s)$ is never an interval of positive length, i.e., each signature produced by f arises at but one point. Q.E.D.

The following corollary is a partial converse to the corollary to Theorem 1; it is given without proof.

MONOTONICITY COROLLARY 4. *Suppose that f and g are saw functions with $SR_f = SR_g$ and suppose that g is signature-distinct. The function h: $[0, 1) \to [0, 1)$ defined by $h(x) =$ the unique y in $[0, 1)$ for which $\text{sig}_g(y) = \text{sig}_f(x)$ has the following properties:*

(a) *$h(0) = 0$, $h($breakpoint of $f) =$ breakpoint of g*

(b) *h is monotone increasing*

(c) *h is onto $[0, 1)$*

(d) *h is continuous*

(e) *h is a homeomorphism* iff *f is signature-distinct*

(f) $hf = gh$; *if f is signature-distinct,* $g = hfh^{-1}$.

This naturally leads to:

THEOREM 3. *Let f and g be signature-distinct saws. We have* $SR_f = SR_g$ *iff f and g are isomorphic.*

Proof. By the corollary to Theorem 1 and Monotonicity Corollary 4. Q.E.D.

In summary the simple concept of signature repertoire contains enough information to determine the isomorphism class of signature-distinct saw functions in an instrinsic fashion. Consequently it behooves us to understand more thoroughly the content and structure of signature repertoires. This is the central topic of the next section.

III. EXTENDED SAWS

Motivated by the desire to determine the content and structure of arbitrary saw functions' signature repertoires we hereby introduce a modified class of transition mappings (the class of extended saws) for which this determination can be carried out with minimal technical difficulty. Once the determination has been made (in Subsection B below) we shall return to derive analogous results for signature-distinct ordinary saw functions (in Subsection C). The latter development will lean heavily on extended saw theory. The authors know no shorter path.

As their definitions will instantly reveal, extended saws are not really functions in the modern mathematical sense of the word, because they are multiply-defined at one point b of their domain $[0, 1]$. Strictly speaking, extended saws are relations between $[0, 1]$ and itself. Unfortunately, the rigorous "relation" terminology obscures the more pertinent understanding that extended saws are but another way of dealing with the discontinuity in the graph of a function. We will sidestep this terminological issue in what follows by simply using the term "extended saws," as opposed to "extended saw functions" or "saw relations."

A. *Elementary Material*

The following paragraphs introduce extended saws, orbits, and signatures in a fashion analogous to the definitions for ordinary saw functions given in the previous section. By reason of said analogy many of the straightforward proofs below are omitted for the sake of brevity.

DEFINITION 12. An extended saw F is a multiple-valued mapping from $[0, 1]$ to $[0, 1]$ having the following properties:

(a) the multiple-valuedness occurs at a single point b in $(0, 1)$, where $F(b)$ is both 0 and 1. We shall call b the breakpoint of F.

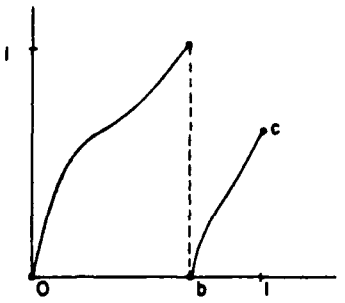


FIG. 4. An extended saw.

(b) F restricted to $[0, b)$ is continuous, strictly monotone increasing, and onto $[0, 1)$.

(c) F restricted to $(b, 1]$ is continuous, strictly monotone increasing, and onto $(0, c]$ for some c obeying $0 < c \leqslant 1$.

Figure 4 shows the graph of a typical extended saw. Clearly given an ordinary saw function f an extended saw results by annexing the points $(b, 1)$ and $(1, c)$ to the graph of f and all possible extended saws arise in this fashion. If f is a saw and F an extended saw so derived, we call F the extension of f and f the underlying saw of F. Referring to Definition 11, note that F_0 and F_1 represent the left and right halves of F and that each function is strictly monotone increasing on its domain.

DEFINITION 13. Let F be an arbitrary extended saw, x in $[0, 1]$. Define $x_0 x_1 x_2 \ldots$ (where x_i in $[0, 1]$ for all i), to be an orbit for x under F iff $x_0 = x$ and $x_{i+1} = F(x_i)$ for all i. Note that if $x_i = b$ then x_{i+1} can be either 0 or 1, so some points x have more than one orbit under this definition. Writing $\text{orb}_F(x)$ as the set of all possible orbits of x, we say x is regular if $\text{orb}_F(x)$ is a singleton set; otherwise we call x irregular.

All irregular points are representable as $F^{-k}(b)$ for some nonnegative integer k. Since $F^{-k}(b)$ contains at most $2^{**}k$ elements, the totality of irregular points for a given F is finite or countably infinite. In fact, as a moment's thought will reveal, it is countably infinite.

DEFINITION 14. *Let F, x be as above, and let $x_0 x_1 x_2 \ldots$* be an orbit for x. The signature of x associated with this orbit is the binary sequence $s_0 s_1 s_2 \ldots$, for which $s_i = 0$ if $x_i < b$; $s_i = 1$ if $x_i > b$ and if $x_i = b$ then $s_i = 0$ if $x_{i+1} = 1$, $s_i = 1$ if $x_{i+1} = 0$.

Defining $\text{sig}_F(x)$ as the set of signatures of x so associated with the elements of $\text{orb}_F(x)$ we note the following relationship with extended saw signatures and ordinary saw function signatures: Let f be an ordinary saw function and let F be its extension. For any x in $[0, 1)$, $\text{sig}_f(x)$ is an element of $\text{sig}_F(x)$. If $\text{orb}_f(x)$ includes b,

however, then $\text{sig}_F(x)$ contains additional signatures corresponding to the extra orbit(s) arising from the point b.

THEOREM 4 (Monotonicity of signatures for extended saws). *Let F be an arbitrary extended saw, x and y points in $[0, 1]$. Let $s_0 s_1 s_2 \ldots$ be an element of $\text{sig}_F(x)$, $t_0 t_1 t_2 \ldots$ an element of $\text{sig}_F(y)$. Then $x < y$ implies $s_0 s_1 s_2 \ldots \leqslant t_0 t_1 t_2 \ldots$.*

DEFINITION 15. The signature repertoire SR_F of an extended saw F is the set $\{s \mid s \text{ is in } \text{sig}_F(x) \text{ for some } x \text{ in } [0, 1]\}$. For any s in S (Definition 6) the set $p_F(s)$ is equal to $\{x \mid x \text{ in } [0, 1] \text{ and } s \text{ is in } \text{sig}_F(x)\}$.

The above definitions are a straightforward generalization of those of the preceding section. Note that an immediate consequence of Theorem 4 is that, for any s, $p_F(s)$ is an interval. It is non-empty iff s is in SR_F.

B. *Topological Considerations*

The purpose of the above discussion has been to introduce extended saws on an equal footing with their ordinary saw counterparts. Recalling that the ultimate purpose of this introduction was to expedite analysis of signature repertoire content, we now proceed to derive a fundamental result: SR_F is completely determined by $\text{sig}_F(1)$! The demonstration proceeds in two stages: (a) showing that the initial portions of strings in SR_F are determined by $\text{sig}_F(1)$, and (b) showing that initial-portion repertoires determine SR_F via a limiting process. Some uniformity of notation is called for:

DEFINITION 16. Let S^k denote the set of all length k strings of 0s and 1s; S^k thus contains 2^k members. Given s in S let $s[0, k)$ denote the k-bit string $s_0 s_1 \ldots s_{k-1}$, $k > 0$. Similarly define $s[0, k]$ to include s_k. Given an extended saw F and a point x in $[0, 1]$, define the k-signatures of x, $\text{sig}_k^F(x)$ via $\text{sig}_k^F(x) = \{s[0, k) \mid s \text{ is in } \text{sig}_F(x)\}$. Let SR_k^F, the k-signature repertoire of F, denote all length k strings which arise in this fashion. Thus SR_F^k is a set of strings, analogous to SR_F. If s is a string of length k or greater, let $p_F^k(s)$ denote the set of all x such that $s[0, k)$ is in $\text{sig}_F^k(x)$. Note that $p_F^k(s)$ is empty iff $s[0, k)$ is not in SR_F^k. View S^k as lexicographically ordered by $<$ as S is.

LEMMA 1. *For any extended saw F and any x in $[0, 1]$, $\text{sig}_F(x)$ is closed as a subset of S in the topology* TOP (*Definition* 6).

Proof. Choose any sequence of signatures s^i in $\text{sig}_F(x)$ which coverges in TOP to some sequence s in S. If s is not in $\text{sig}_F(x)$ then it fails to be a signature for x due to a fault in its first k terms, for some value of k. By the nature of covergence in TOP, then, for i sufficiently large, the s^i themselves are not signatures of x. Contradiction. So s is in $\text{sig}_F(x)$. Q.E.D.

By virtue of Lemma 1, we can make the following definitions:

DEFINITION 17. Denote $\sup(\mathrm{sig}_F(x))$ by $\mathrm{rs}_F(x)$, called the right signature of x, and denote $\inf(\mathrm{sig}_F(x))$ by $\mathrm{ls}_F(x)$, the left signature of x. By Lemma 1, $\mathrm{rs}_F(x)$ and $\mathrm{ls}_F(x)$ are in $\mathrm{sig}_F(x)$.

Note that if x is regular, it must be that $\mathrm{sig}_F(x) = \{\mathrm{ls}_F(x)\} = \{\mathrm{rs}_F(x)\}$. Furthermore, the above "lopping properties" hold for left and right signatures:

$$\mathrm{lop}(\mathrm{rs}_F(x)) = \mathrm{rs}_F(F(x)) \qquad \text{for } x \text{ not equal to } b,$$

$$\mathrm{lop}(\mathrm{ls}_F(x)) = \mathrm{ls}_F(F(x)) \qquad \text{for } x \text{ not equal to } b,$$

$$\mathrm{lop}(\mathrm{rs}_F(b)) = 0000\ldots,$$

$$\mathrm{lop}(\mathrm{ls}_F(b)) = \mathrm{ls}_F(1).$$

Note that s and t in S with $s \leqslant t$ implies $s[0, k) \leqslant t[0, k)$. The principle of monotonicity of k-signatures immediately follows: s in $\mathrm{sig}_F^k(x)$ and t in $\mathrm{sig}_F^k(y)$ and $x < y$ implies $s \leqslant t$. It is immediate that if x and y share a k-signature s then every z between x and y must share this signature and this be the only signature for z. Hence $p_F^k(s)$ for any s in SR_F^k is an interval. In fact, it is a closed interval, as the following lemma demonstrates. We omit its straightforward proof which simply relies upon the closedness of the intervals $[0, b]$ and $[b, 1]$.

LEMMA 2. *For any s in SR_F^k the point set $p_F^k(s)$ is a closed subset of $[0, 1]$.*

Proof. The proof is straightforward and is omitted.

What is the relationship between the $p_F^k(s)$ and $p_F(s)$?

LEMMA 3. *$p_F(s) = \{x \mid x$ is in $p_F^k(s)$ for all $k\}$ for any s in SR_F, i.e., $p_F(s)$ is the intersection over k of the $p_F^k(s)$.*

Proof. From basic definitions it is clear that for any s and for any k, $p_F(s)$ is a subset of $p_F^k(s)$. Conversely, suppose that x is in $p_F^k(s)$ for all values of k, for a given s. Then there must exist a sequence of elements of $\mathrm{sig}_F(x)$ $s^1, s^2, s^3, \ldots$, for which $s^i[0, i)$ agrees with $s[0, i)$ for all i. But this means that in the topology TOP, $\{s^i\}$ converges to s. Since $\mathrm{sig}_F(x)$ is closed (Lemma 1) it must be the case then that s is in $\mathrm{sig}_F(x)$ too, i.e., x is in $p_F(s)$. Q.E.D.

COROLLARY. *$p_F(s)$ is a closed interval.*

Note. Any function g defined on points x may be extended to a function g defined on sets of points X by writing $g(X) = \{g(x) \mid x$ is in X and x is in the domain of $g\}$. We shall use this convention below in referring to sets $F_0(X)$ and $F_1(X)$ for various subsets X of $[0, 1]$. We similarly will talk about $\mathrm{sig}_F(X)$ and $\mathrm{sig}_F^k(X)$ for subsets X of $[0, 1]$ and to simplify notation will write $g[p, q]$ instead of $g([p, q])$ when the sets X in question are intervals.

The next Lemma shows how $(k+1)$-signatures are related to k-signatures; its proof is straightforward and is omitted.

LEMMA 4. *For any subset X of $[0, 1]$, and $k \geq 1$*

$$\mathrm{sig}_F^{k+1}(X) = 0 \cdot \mathrm{sig}_F^k(F_0(X)) \cup 1 \cdot \mathrm{sig}_F^k(F_1(X)),$$

where the dots denote string concatenation.

We also omit the proof of

LEMMA 5. *Given arbitrary x and y obeying $0 \leq x \leq y \leq 1$, for any $k \geq 1$,*

$$\mathrm{sig}_F^k[x, y] = \mathrm{SR}_F^k \wedge [\mathrm{ls}_F^k(x), \mathrm{rs}_F^k(y)],$$

where $\wedge$ denotes intersection, and the closed interval shown is an interval in S^k.

The Recursion Theorem

All of the above culminates in what may be regarded as one of the fundamental results of this paper.

THEOREM 5 (Recursion theorem). *Let F be an arbitrary extended saw. Then for all $k \geq 0$*

$$\mathrm{SR}_F^{k+1} = (0 \cdot \mathrm{SR}_F^k) \cup (1 \cdot \mathrm{SR}_F^k \wedge [10^k, \mathrm{rs}_F^{k+1}(1)]).$$

Proof. Recalling that $F(1) = c$ we have

$$\begin{aligned}
\mathrm{SR}_F^{k+1} &= \mathrm{sig}_F^{k+1}[0, 1] && \text{(by definition)} \\
&= 0 \cdot \mathrm{sig}_F^k(F_0[0, 1]) \cup 1 \cdot \mathrm{sig}_F^k(F_1[0, 1]) && \text{(by Lemma 4)} \\
&= 0 \cdot \mathrm{sig}_F^k[0, t] \cup 1 \cdot \mathrm{sig}_F^k[0, c] && \text{(by direct calculation)} \\
&= 0 \cdot (\mathrm{SR}_F^k \wedge [\mathrm{ls}_F^k(0), \mathrm{rs}_F^k(1)]) \\
&\quad \cup 1 \cdot (\mathrm{SF}_F^k \wedge [\mathrm{ls}_F^k(0), \mathrm{rs}_F^k(c)]) && \text{(by Lemma 5).}
\end{aligned}$$

Since SR_F^k is a subset of $[\mathrm{ls}_F^k(0), \mathrm{rs}_F^k(1)]$ we thus have

$$\begin{aligned}
\mathrm{SR}_F^{k+1} &= 0 \cdot \mathrm{SR}_F^k \cup 1 \cdot (\mathrm{SR}_F^k \wedge [\mathrm{ls}_F^k(0), \mathrm{rs}_F^k(c)]) \\
&= (0 \cdot \mathrm{SR}_F^k) \cup (1 \cdot \mathrm{SR}_F^k \wedge [10^k, \mathrm{rs}_F^{k+1}(1)])
\end{aligned}$$

(by direct calculation). Q.E.D.

We always have, or course, that $\mathrm{SR}_F^1 = \{0, 1\}$; by mathematical induction employing the recursion of Theorem 5, then, it follows that all finite length signatures (initial portions of elements of SR_F) are determined by $\mathrm{rs}_F(1)$ (and hence by $\mathrm{sig}_F(1)$). It will now be demonstrated that SR_F itself is determined by $\mathrm{sig}_F(1)$ (Theorems 6 and 7.)

DEFINITION 18. Given any sequence T^* of subsets of S^* define lim T^* to be the subset T of S such that t is in T iff $t[0, k)$ is in T^* for every k.

THEOREM 6. $SR_F = \lim SR_F^k$.

Proof. Clearly SR_F is a subset of lim SR_F^k, for if s is in SR_F then for all k, $s[0, k)$ is in SR_F^k by the definition of SR_F^k. Conversely, suppose that s is in lim SR_F^k. Then $s[0, k)$ is in SR_F^k for all k, and thus, for all k, $p_F^k(s)$ is non-empty. By Lemma 3, $p_F(s)$ is the intersection of $p_F^k(s)$, which themselves form a nested sequence of closed intervals. By elementary analysis, then, $p_F(s)$ is non-empty. Hence, s is in SR_F. Q.E.D.

Finally, we have the following string-theoretic criterion for membership in SR_F, based entirely upon $rs_F(1)$:

THEOREM 7. *Let F be an arbitrary extended saw. Then*

$$SR_F = \{s \mid lop^k(s) \leqslant rs_F(1) \text{ for all } k\}.$$

Proof. If s is in SR_F then for all k, $lop^k(s)$ is in SR_F. $rs_F(1)$ is the maximal member of SR_F by monotonicity of signatures. Conversely, we use a method attributable to Kwankam [6]:

Given s in S with $lop^k(s) \leqslant rs_F(1)$ for all k, we must show that s is in SR_F. By Theorem 6 it suffices to show that $s[0, k)$ is in SR_F^k for all k. We do so by induction on k. We find it easier to prove by induction a more general result. Letting $s[j, j+k]$ denote the finite sequence $s_j s_{j+1} \cdots s_{j+k}$, it will be shown by induction on k that $s[j, j+k]$ is in SR_F^{k+1} for $j = 0, 1, 2, \ldots$. That is, we shall establish that all contiguous length $(k+1)$ substrings of s are $(k+1)$-signatures of F, rather than just concentrating on initial substrings only.

Basis: for $k = 0$, $s[j, j+k]$ is either 0 or 1, but $SR_F^1 = \{0, 1\}$.

Induction step: Assume that $s[i, i+k]$ is a $(k+1)$-signature for all i. Consider for a given j, $s[j, j+k+1]$. Applying the recursion theorem we argue as follows: if $s[j, j+k+1]$ arises from $s[j+1, j+k+1]$ by prefixing a 0, it is immediate that $s[j, j+k+1]$ is a $(k+2)$-signature. If it arises from $s[j+1, j+k+1]$ by prefixing a 1 then we note that it will be a $(k+2)$-signature if it is $\leqslant rs_F^{k+2}(1)$. But this is true from our theorem hypothesis, since $lop^j(s) \leqslant rs_F(1)$. Q.E.D.

C. *The Signature-Distinct Case*

The general results of the previous two subsections will be applied here to produce a detailed theory of signature repertoire content for signature-distinct saw functions. In particular a theorem for determining signature repertoire content analogous to Theorem 7 will be established. Most of the results on signature-distinct saw functions f recorded here will be derived from properties of the extension F of f. Accordingly we will cite some results relating the two (F and f) in greater detail. The proofs are straightforward and are omitted.

LEMMA 6. *Let f be an arbitrary saw and F its extension:*

(a) *Let $x_0 x_1 x_2 \ldots$ be an F-orbit. Then $x_0 x_1 x_2 \ldots$ is an f-orbit* iff *no x_i is equal to* 1.

(b) *Let $x_0 x_1 x_2 \ldots$ be an F-orbit and $s_0 s_1 s_2 \ldots$ the corresponding F-signature. If $x_0 x_1 x_2 \ldots$ is also an f-orbit, then $s_0 s_1 s_2 \ldots$ is also an f-signature (and is the signature associated with this orbit).*

(c) *If $s_0 s_1 s_2 \ldots$ is an f-signature (of the point x, say) then $s_0 s_1 s_2 \ldots = \mathrm{rs}_F(x)$.*

DEFINITION 19. An extended saw F is said to be signature-distinct iff whenever x is different from y then $\mathrm{sig}_F(x)$ and $\mathrm{sig}_F(y)$ are disjoint.

LEMMA 7. *Let f be an arbitrary saw function and F its extension. Then f is signature-distinct iff F is.*

THEOREM 8. (Discriminant theorem). *Let f be a signature-distinct saw function and let F be its extension. Then*

$$\mathrm{SR}_f = \{s \mid \mathrm{lop}^k(s) < \mathrm{ls}_F(1) \text{ for all } k > 0\}.$$

Proof. Let T denote the set of strings on the right of the above equation. We first show that SR_f is a subset of T. If s is in SR_f then there exists an x in $[0, 1)$ such that $s = \mathrm{rs}_F(x)$, by Lemma 6. We know that $f^k(x) < 1$ for all k. Hence by monotonicity, $\mathrm{lop}^k(s) \leqslant \mathrm{ls}_F(1)$ for all k. Since F is signature-distinct (Lemma 7) and $x < 1$, it follows that $\mathrm{lop}^k(s) < \mathrm{ls}_F(1)$ for all k. Hence SR_f is a subset of T.

Conversely, suppose s is in T. By Theorem 7, s is in SR_F. Thus there exists x_0 in $[0, 1]$ with orbit $x_0 x_1 x_2 \ldots$, such that s is an F-signature for x_0 arising from this orbit. If this orbit contains the term $x_k = 1$ then $\mathrm{lop}^k(s)$ is a signature of 1 arising from the F-orbit $x_k x_{k+1} x_{k+2} \ldots$; hence $\mathrm{lop}^k(s) \geqslant \mathrm{ls}_F(1)$, contradicting the definition of T. Thus the orbit $x_0 x_1 x_2 \ldots$ contains no $x_k = 1$ and so is an f-orbit by Lemma 6. So again by Lemma 6, s is an f-signature. Q.E.D.

DEFINITION 20. Let f be an arbitrary saw function. Any string d in S for which $\mathrm{SR}_f = \{s \mid \mathrm{lop}^k(s) < d \text{ for all } k \geqslant 0\}$ will be called a discriminant for f and will be denoted by discrim(f).

Thus, a discriminant for f is any string by which one may readily discriminate between members and non-members based upon a "lop test." Theorem 8 demonstrates that $\mathrm{ls}_F(1)$ is a discriminant for a signature-distinct saw function f. The discussion below will show that this is indeed the only discriminant for f, justifying the functional notation discrim(f).

DEFINITION 21. A sequence $s_0 s_1 s_2 \ldots$ in S is said to be lop-maximal if for all $k \geqslant 0$, $\mathrm{lop}^k(s) \leqslant s$.

(Note that non-lop-maximal sequences exist, e.g., 010111..., as well as lop-maximal ones, e.g., 11000....)

THEOREM 9. *For any extended saw F, $\mathrm{ls}_F(1)$ is lop-maximal.*

Proof. Recall that the lop of the left-signature of x produces the left-signature of $F(x)$ when x is not equal to b, and of 1 when x equals b; thus lopping takes left-signatures into left-signatures. By montonicity of signatures $\mathrm{ls}_F(1)$ is the largest possible left-signature of F; the theorem immediately follows. Q.E.D.

THEOREM 10 (Continuity of signatures). *Let F be an arbitrary extended saw. If $x_0\, x_1\, x_2 \ldots$ is a sequence of points from $[0, 1]$ converging to some point x of $[0, 1]$ from the left, with no x_i equal to x, then for any choice of signatures $s^0, s^1, \ldots$, for the x_i the limit $\lim s^i$ in the $<$ ordering on S exists and is equal to $\mathrm{ls}_F(x)$. Similarly if $y_0\, y_1\, y_2 \ldots$ converges to y strictly from the right, then for any choice of signatures $t^0, t^1, \ldots$, for the y_i the limit $\lim t^i$ exists and equals $\mathrm{rs}_F(y)$.*

Proof. The proof is straightforward and will be omitted.

COROLLARY 1. *Let f be an arbitrary saw and F its extension. Then* $\sup \mathrm{SR}_f = \mathrm{ls}_F(1)$.

Proof. By Lemma 6(c), $\mathrm{SR}_f = \{s \mid s = \mathrm{rs}_F(x) \text{ for } x \text{ in } [0, 1)\}$. By monotonicity of F-signatures, it is immediate that $\mathrm{ls}_F(1)$ is an upper bound for SR_f. By continuity of signatures if $\{x_i\}$, x_i in $(0, 1)$, converges to 1 from the left, $\mathrm{rs}_F(x_i)$ converges to $\mathrm{ls}_F(1)$. Hence a sequence of elements of SR_f converges to an upper bound for it; this upper bound must then be the sup. Q.E.D.

COROLLARY 2. *Let f be a signature-distinct saw-function. Then the discriminant for f is unique.*

Proof. Let $d = \sup \mathrm{SR}_f$. By Corollary 1, d is equal to $\mathrm{ls}_F(1)$ and is thus a discriminant for f; since d is certainly not less than itself, d is not in SR_f. Suppose there were a second discriminant d' for f. By the very definition of discriminant d' is an upper bound for SR_f. Hence $d' > d$. By the lop-maximality of d (Theorem 9), d thus satisfies the d'-discriminant criterion for being a member of SR_f. Contradiction. Conclusion: there is only one discriminant for f. Q.E.D.

Can signature repertoire content for a signature-distinct function be narrowed down any further? In terms of the present development this amounts to asking whether discrim(f) can be an arbitrary lop-maximal sequence or whether there are further inherent restrictions on its structure. The following key result yields yet another restriction on the form of discrim(f).

THEOREM 11. *If F is a signature-distinct extended saw then* discrim(f) *contains infinitely many* 1s.

Proof. By contradiction, suppose discrim(f) contains only finitely many 1s (it always contains at least 1 since discrim(f) = $ls_F(1)$ and hence the first bit of discrim(f) is a 1). Thus it is of the form

$$\text{discrim}(f) = s_0\, s_1\, s_2 \ldots s_k\, 0\, 0\, 0 \ldots,$$

with $s_k = 1$ for some integer $k \geq 0$. Since discrim(f) = $ls_F(1)$, it follows from the lopping properties for extended saws that 100... is a signature of $F^*(1)$. Now 100... is also the right signature of b, the breakpoint of F, and, thus, by the signature-distinctness of F, it must be that $F^*(1)$ is equal to b. Since b can map to 1, it follows that 1 has an orbit that periodically returns to itself. Since this orbit always uses F_0 (as opposed to F_1) whenever there is a choice it follows that the signature of this orbit is $ls_F(1)$, the least signature of 1. Conclusion: $ls_F(1)$ is periodic and thus contains infinitely many 1s—a contradiction. Discrim(f) must really contain infinitely many 1s. Q.E.D.

We can go no further is reducing the possibilities for discrim(f): any sequence of 0s and 1s which is lop-maximal and which contains infinitely many 1s can really arise as the discriminant of a signature-distinct saw function. This requires demonstration of course! The next section will substantiate this assertion by actually constructing the requisite function f. Remarkably the constructed f will always be piecewise-linear with parallel pieces, yielding as a bonus the theory of piecewise-linear prototypes of Section V.

IV. Real-Radix Number Systems

In this section we introduce a simple generalization of standard positional number notations, of which the binary and decimal notations are familiar examples. The essence of the generalization consists in allowing the radix r of the representation to be an arbitrary real number $r > 1$, as opposed to an arbitrary integer greater than 1. Although our development could be carried out with full $r > 1$ generality, we shall insist that r be confined to the range $(1, 2]$. This assumption allows us to employ the results of Sections II and III to develop rapidly the properties of these number systems. The goal of this development is reached in Theorem 15: any lop-maximal sequence with infinitely many 1s is a discriminant.

The proofs of Theorems 12 and 13 in this section are straightforward and will be omitted for lack of space.

Definition 22. Given any real number r in $(1, 2]$ and any real number x in $[0, 1)$ define the radix r representation of x as the sequence $q_0\, q_1 \ldots$ of integers given by the "Radix r Representation Algorithm":

Step 1. Set $s = 1/r$, $x_0 = x$, and $i = 0$.

Step 2. Divide x_i by s^{i+1}; let q_i be the integer valued quotient and let x^{i+1} be the non-negative remainder. Thus

$$x_i = q_i s^{i+1} + x_{i+1} \qquad \text{with} \quad 0 \leqslant x_{i+1} < s^{i+1}.$$

Step 3. Set $i = i + 1$ and go back to Step 2.

THEOREM 12. *Let r, x, s be as in Definition 22, and let $q_0 q_1 \ldots$ be the sequence derived by the above algorithm. Then $q_0 q_1 \ldots$ has the following properties:*

(a) *each q_i is either 0 or 1,*

(b) $x = q_0 * s^1 + q_1 * s^2 + \ldots,$

(c) $x_k = q_k * s^{k+1} + q_{k+1} * s^{k+2} + \ldots,$

(d) $s^k > q_k * s^{k+1} + q_{k+1} * s^{k+2} + \ldots.$

The Radix r Representation Algorithm insures that every x in $[0, 1)$ has a well-defined representation which, by (b), faithfully represents x. Statement (d) implies that not every sequence of 0s and 1s actually arises as the representation of some quantity in $[0, 1)$. Indeed for any r in $(1, 2]$ the sequence consisting all of 1s fails to satisfy the inequality. It is important in the sequel to know exactly which sequences of 0s and 1s actually arise as radix r expansions of numbers x in the interval $[0, 1)$. Condition (d) turns out to be sufficient as well.

THEOREM 13. *A sequence $q_0 q_1 \ldots$ of 0s and 1s is the radix r expansion of some x in $[0, 1)$ if for all $k \geqslant 0$*

$$s^k > q_k * s^{k+1} + q_{k+1} * s^{k+2} + \ldots.$$

The x yielding this expansion is unique and is given by $q_0 * s + q_1 * s^2 + \ldots$.

DEFINITION 23. Given r in $(1, 2]$ define RRNS(r), the real-radix number system of radix r, to be the collection of all strings output by the Radix r Representation Algorithm in response to inputs selected from $[0, 1)$. Define E_r: $[0, 1) \to$ RRNS(r) as the function which associates with each x in $[0, 1)$ its expansion in RRNS(r). Define V_r: $S \to$ the real numbers to be the function which associates with any string $s_0 s_1 s_2 \ldots$, the real value $x = s_0/r + s_1/r^2 + \ldots$.

Note that V_r is a continuous function for each r and that V_r restricted to RRNS(r) and E_r are inverse mappings. Also note that we may summarize Theorems 12 and 13 by stating that RRNS(r) $= \{q \mid V_r(\text{lop}^k(q)) < 1$ for all $k \geqslant 0\}$. The alert reader will immediately observe the formal similarity between this representation of RRNS(r) and the representation of SR_f as given by Theorem 8. Could there be a connection?

DEFINITION 24. For r in $(1, 2]$ define the mapping f_r: $[0, 1) \to [0, 1)$ as follows:

given x in $[0, 1)$ with radix r expansion $q_0 q_1 q_2 \ldots$, take $f_r(x) = q_1/r + q_2/r^2 + \ldots$. Thus $f_r(x)$ is obtained by "lopping" the radix r expansion of x.

THEOREM 14 (Link between number systems and saw functions). *f_r is a signature-distinct saw function with breakpoint $1/r$. For any x in $[0, 1)$ $\mathrm{sig}_{f_r}(x)$ is equal to $E_r(x)$.*

Proof. We claim that *$f_r(x)$ is given by the formula*

$$f_r(x) = rx \qquad \text{if } x \text{ is in } [0, 1/r)$$
$$= rx - 1 \qquad \text{if } x \text{ is in } [1/r, 1)$$

for if $x < 1/r$ then q_0 (in Definition 24) must be 0 by the expansion algorithm and consequently $f_r(x)$ is merely r times x. If $x \geq 1/r$ then $q_0 = 1$ and so $f_r(x)$ is $rx - 1$. Figure 5 shows the graph of $f_r(x)$. It is clearly a saw function with breakpoint $1/r$. Since its pieces are parallel with slope r, which is greater than 1, f_r is pse and hence signature-distinct by the Monotonicity Corollary 3 of Section II. The lopped sequence $q_1 q_2 \ldots$ clearly obeys the inequality of Theorem 13 (since $q_0 q_1 \ldots$ does) and thus is the radix r expansion for $f_r(x)$. So, by induction, $E_r(f^k(x)) = \mathrm{lop}^k(E_r(x))$ for all k. Hence the first term of $\mathrm{lop}^k(E_r(x))$ tells whether or not $f^k(x)$ is less than $1/r$. That is, E_r is equal to sig_{f_r}. Q.E.D.

COROLLARY 1. *E_r is strictly monotone increasing.*

Proof. This follows immediately from monotonicity of signatures applied to f_r, the signature-distinctness of f_r, and the equation $E_r = \mathrm{sig}_{f_r}$. Q.E.D.

COROLLARY 2. $\mathrm{RRNS}(r) = \{q \mid \mathrm{lop}^k(q) < \mathrm{discrim}(f_r)\}$; *thus there exists a discriminant sequence for determining membership in* $\mathrm{RRNS}(r)$.

Proof. By Theorem 8 of Section III and the fact that $\mathrm{RRNS}(r) = SR_{f_r}$. Q.E.D.

COROLLARY 3. *$V_r(\mathrm{discrim}(f_r)) = 1$ and, for all $k \geq 0$, $V_r(\mathrm{lop}^k(\mathrm{discrim}(f_r))) \leq 1$.*

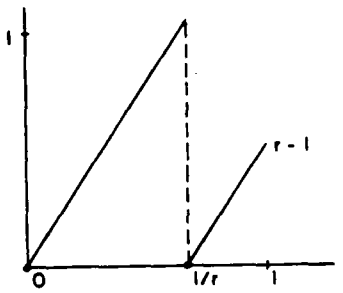


FIG. 5. The prototype saw function f_r.

Proof. Let x_i be a sequence of points of $[0, 1)$ converging to 1. By continuity of signatures (Theorem 10), $\mathrm{rs}_{F_r}(x_i)$ converges to $\mathrm{ls}_{F_r}(1)$, where F_r is the extension of f_r. Consequently $\mathrm{lop}^k(\mathrm{rs}_{F_r}(x_i))$ converges to $\mathrm{lop}^k(\mathrm{ls}_{F_r}(1))$ for all k. Hence, by the continuity of V_r, the numerical sequence having as its ith term $V_r(\mathrm{lop}^k(\mathrm{rs}_{F_r}(x_i)))$ converges to $V_r(\mathrm{lop}^k(\mathrm{ls}_{F_r}(1)))$ for every k. For $k=0$ the indicated numerical sequence is just x_i, since for $x<1$, $\mathrm{rs}_{F_r}(x)$ is equal to $\mathrm{sig}_{f_r}(x)$ (Lemma 6(c)) and we have just shown that signatures under f_r and notations radix r coincide. Because the x_i converge to 1, and because V_r (any signature of f_r) is <1, the relations in the corollary statement immediately follow. Q.E.D.

We are finally in a position to demonstrate that any lop-maximal sequence with an infinite number of 1s is the discriminant of a signature-distinct saw function. We show

THEOREM 15. *Let $d=d_0\,d_1\,d_2\ldots$ be lop-maximal with infinitely many 1s. Then there exists a unique r in $(1, 2]$ such that $d=\mathrm{discrim}(f_r)$.*

Proof of Uniqueness. If $d=\mathrm{discrim}(f_r)$ then $V_r(d)=1$ by Corollary 3 above. Hence r is a root of $g(x)=1$, where $g(x)=d_0/x+d_1/x^2+\ldots$. The function $g(x)$ is strictly monotone decreasing on the nonnegative reals and consequently has at most one root in the region of interest. Hence if r exists it is unique.

Proof of Existence. Let $R=\{r \mid d \text{ is in } \mathrm{RRNS}(r)\}$. If $d=111\ldots$, then $d=\mathrm{discrim}(f_2)$ and we are done. If d is not equal to $111\ldots$, then no tail of d can be $111\ldots$, either, since d is lop-maximal. Hence $V_2(\mathrm{lop}^k(d))<1$ for all k. Consequently d is in RRNS(2) and therefore 2 is in R. Thus R is non-empty and inf R exists. Call it r. We claim that $d=\mathrm{discrim}(f_r)$. We do this in two steps. We first argue that $V_r(d)$ is equal to 1 as follows:

(a) for any $t>r$ t is in R; hence d is in $\mathrm{RRNS}(t)$ and $V_t(d)<1$;

(b) since d is lop-maximal and V_t is strictly monotone increasing on $\mathrm{RRNS}(t)$ (being the inverse of E_t) $V_t(\mathrm{lop}^k(d))<V_t(d)$ for all $k\geqslant 0$;

(c) from (a) and (b) and the continuity of V_x as a function of x $V_r(\mathrm{lop}^k(d))\leqslant V_r(d)\leqslant 1$ for all $k\geqslant 0$;

(d) if $V_r(d)$ is not equal to 1, consider the family of maps $\{g_k\colon (1, 2]\to \text{reals}\}$ given by $g_k(x)=V_x(\mathrm{lop}^k(d))$. This family of maps is equicontinuous. If $V_r(d)<1$ then there exists $t<r$ so that for all $k\geqslant 0$, $V_t(\mathrm{lop}^k(d))<1$, i.e., t is in R. This contradicts the definition of r. Thus $V_r(d)$ equals 1.

We now show that $d=\mathrm{discrim}(f_r)$. Suppose $d<\mathrm{discrim}(f_r)$. Then $\mathrm{lop}^k(d)<\mathrm{discrim}(f_r)$ by the lop-maximality of d, and this implies d is in $\mathrm{RRNS}(r)$, contradicting the fact that $V_r(d)=1$.

If $d>\mathrm{discrim}(f_r)$, we look for the smallest position k for which these two strings differ. Consider $\mathrm{lop}^k(d)$. It has an initial term of 1 and hence has value $>1/r$ (strictly greater due the presence of infinitely many 1s in d. Similarly $\mathrm{lop}^k(\mathrm{discrim}(f_r))$ must begin with a 0 and it follows from Corollary 3 above then that

$V_r(\text{lop}^k(\text{discrim}(f_r)))$ has value $\leq 1/r$. Herein lies the contradiction. $V_r(\text{discrim}(f_r))$ and $V_r(d)$ both have values of 1; hence these two tails must have the same value.

Consequently, we have neither $d < \text{discrim}(f_r)$ or $d > \text{discrim}(f_r)$, so $d = \text{discrim}(f_r)$. Q.E.D.

V. Theory of Prototypes for Signature-Distinct Saws

The goal towards which the foregoing material has been aiming is that every signature-distinct saw function is isomorphic to one of the functions f_r introduced in Section IV. We are now in a position to formally demonstrate this result.

THEOREM 16 (Prototype theorem). *Let f be a signature-distinct saw function. Then there exists an r in $(1, 2]$ such that f is isomorphic to the piecewise-linear, parallel piece saw function f_r.*

Proof. By Theorem 9, $\text{discrim}(f)$ is lop-maximal; by Theorem 11 it has infinitely many 1s. Hence by Theorem 15 there exists a unique r in $(1, 2]$ for which $\text{discrim}(f) = \text{discrim}(f_r)$. Hence by Theorem 8, $\text{SR}_f = \text{SR}_{f_r}$, and, by Theorem 3, f *and* f_r are isomorphic. Q.E.D.

THEOREM 17. *The following are equivalent statements about signature-distinct saw functions f_1 and f_2 and their respective prototypes f_{r_1} and f_{r_2}:*

(a) $\text{discrim}(f_1) < \text{discrim}(f_2)$

(b) SR_{f_1} *is a proper subset of* SR_{f_2}

(c) $\text{RRNS}(r_1)$ *is a proper subset of* $\text{RRNS}(r_2)$

(d) $r_1 < r_2$.

Proof. We show (a) iff (b), (b) iff (c), and then (c) iff (d):

(a) iff (b). Assume (a) is true. Any string with all its lops less than $\text{discrim}(f_1)$ has all its lops less than $\text{discrim}(f_2)$; thus SR_{f_1} is a subset of SR_{f_2} by Theorem 8. It is indeed a proper subset of SR_{f_2} since $\text{discrim}(f_1)$ is certainly not a member of SR_{f_1} (it is not less than itself), but is a member of SR_{f_2} since it is lop-maximal and is less than $\text{discrim}(f_2)$.

Conversely, if $\text{discrim}(f_1) = \text{discrim}(f_2)$, then by Theorem 8, SR_{f_1} is equal to SR_{f_2}. If $\text{discrim}(f_1) > \text{discrim}(f_2)$ then, by repeating the above argument, SR_{f_2} is a proper subset of SR_{f_1}. Thus contradictions arise for either assumption. The conclusion: $\text{discrim}(f_1) < \text{discrim}(f_2)$.

(b) iff (c). By Theorem 14, Corollary 2, note that $\text{RRNS}(r_1) = \text{SR}_{f_{r_1}} = \text{SR}_{f_1}$, and $\text{RRNS}(r_2) = \text{SR}_{f_{r_2}} = \text{SR}_{f_2}$. The result is immediate.

(c) iff (d). Assume that $r_1 < r_2$. Recall the characterization from Theorems 12 and 13 that $\text{RRNS}(r) = \{q \mid V_r(\text{lop}^k(q)) < 1\}$. Now for $r < r'$ it is always true that

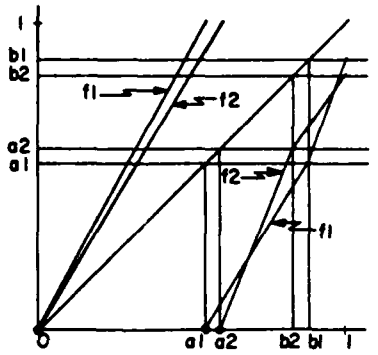


FIG. 6. Isomorphic saw functions.

$u_0/r + u_1/r^2 + \ldots < 1$ implies $u_0/r' + u_1/r'^2 + \ldots < 1$ for any sequence $u_0\, u_1 \ldots$. It is immediate then that RRNS(r_1) is subset of RRNS(r_2). It is a strict subset because the string discrim(f_{r_1}) is not in RRNS(r_1)—by Theorem 14, V_{r_1}(discrim(f_{r_1})) = 1—but is in RRNS(r_2) (since V_{r_1}(lop^k(discrim(f_{r_1}))) $\leqslant 1$ for all k, by Theorem 14, then for all k, V_{r_2}(lop^k(discrim(f_{r_1}))) is < 1, since $r_1 < r_2$).

Conversely, if $r_1 = r_2$ then certainly RRNS(r_1) = RRNS(r_2); if $r_1 > r_2$ then, by the above argument, RRNS(r_2) is a proper subset of RRNS(r_1). It must be then that $r_1 < r_2$. Q.E.D.

The above theorem is significant in that it shows that every signature-distinct saw has a unique prototype, which in turn corresponds to a unique radix. In informal terms the family of prototypes is a 1-parameter family—r being the parameter; as r increases from 1 to 2 the signature repertoire of the prototype increases accordingly. Any signature assumable by f_{r_1} is assumable by f_{r_2} iff $r_1 \leqslant r_2$. This property reflects back to signature- distinct saws; by comparing discrim(f_1) with discrim(f_2) we can determine which saw can "simulate" the other—a rather remarkable result.

As a consequence of this discussion we can immediately conclude that the two saw functions illustrated in Fig. 6 are isomorphic: they are both signature-distinct since they are pse, and their respective extensions share common signatures for 1 (and hence a common left-signature).

VI. CONCLUSIONS

The following results have been derived:

(a) Every saw function has a signature repertoire. Two signature-distinct saw functions are isomorphic iff they have the same repertoire.

(b) For every signature-distinct saw function f there exists a unique string discrim(f) called the discriminant of f, such that an arbitrary string s is in the

signature repertoire SR_f of f iff all "tails" of s, and s itself, are strictly less than discrim(f) in lexicographical order.

(c) discrim(f) may be obtained as either $ls_F(1)$, where F is the extension of f, or as $\sup SR_f$.

(d) The collection of discriminants so arising can be characterized in strictly string-theoretic terms: a string d is the discriminant for some signature-distinct saw function iff it is lop-maximal and contains infinitely many 1s.

(e) Discriminants d are in one-to-one correspondence with radices r chosen from (1, 2]. The correspondence is given by r is the unique positive root of $d_0/x + d_1/x^2 + \ldots = 1$.

(f) Each discriminant arises from a particular 2-piece piecewise-linear saw function. Consequently every signature-distinct saw function is isomorphic to one of these prototype saw functions.

The theory of prototypes has far-reaching implications. The iterative structure of any signature-distinct saw function may be studied by considering the structure of an isomorphic parallel-piece linear saw. Fixed points, periodic orbits, orbital topology, asymptotic behavior, and ergodic properties of orbits—all can be investigated through consideration of these elementary constant-slope, piecewise-linear maps. The simplification effected is indeed remarkable in extent.

Due to the central role played by piecewise-linear functions in this theory, and also due to the ability of piecewise-linear maps to approximate the behavior of more general maps, the iterative behavior of the piecewise-linear family becomes an especially interesting research topic.

Perhaps the most discouraging aspect of the theory expounded here is the number of restrictions placed on the class of systems studied: a 1-dimensional state space; autonomous operation; transition functions which have two continuous and monotone pieces; signature-distinctness. We believe however that the methods employed in this paper will routinely extend to general 2-piece functions ($f(0) = 0$ and so on not required) and also to p-piece functions with $p > 2$. The general areas of non-autonomous systems and of higher-dimensional systems remain totally untouched topics.

A special reference is required for three pioneering works on number notations and prototype theory, works unknown to the authors for more than one year after their own independent work in this area. Everett [2] apparently first investigated the existence of real radix number systems; Parry [10] then thoroughly developed their properties and [11] derived a prototype theory by means of ergodic theory methods. The interested reader is referred to these expositions for an alternative exclusively mathematically oriented view of number systems and prototype theory.

REFERENCES

1. P. COLLET AND J. ECKMANN, "Iterated Maps on the Interval as Dynamical Systems," Birkhauser, Basel, 1980.

2. C. J. EVERETT, Representations of real numbers, *Bull. Amer. Math. Soc.* **52** (1946), 861–869.
3. M. E. KALISKI, Autonomous sequence generation, *Inform. and Control* **26** (1974), 201–218.
4. M. E. KALISKI AND Q. L. KLEIN, Toward a theory of dense orbital behavior in a class of autonomous one-dimensional nonlinear discrete-time systems, *Inform. and Control* **35** (1977), 246–258.
5. Q. L. KLEIN AND M. E. KALISKI, Functional equivalence in a class of autonomous one-dimensional nonlinear discrete-time systems, *Inform. and Control* **42** (1979), 131–147.
6. S. Y. KWANKAM, "A Structure Theory for the Fixed Points of the Iterates of Certain Nonlinear One-Dimensional Functions Defined Over the Unit Interval," Ph.D. thesis, Northeastern University, Boston, Mass.
7. T. Y. LI AND J. A. YORKE, Period three implies chaos, *Amer. Math. Monthly* **82** (1975), 985–992.
8. R. MAY, Simple mathematical models with very complicated dynamics, *Nature* **261** (1976), 459–467.
9. N. METROPOLIS, M. L. STEIN, AND P. R. STEIN, On finite limit sets for transformations of the unit interval, *J. Combin. Theory Ser. A* **15** (1973), 25–44.
10. W. PARRY, On the beta-expansions of real numbers, *Acta Math. Acad. Sci. Hungar.* **11** (1960), 401–416.
11. W. PARRY, Symbolic dynamics and transformations of the unit interval, *Trans. Amer. Math. Soc.* **122** (1966), 368–378.
12. W. RUDIN, "Principles of Mathematical Analysis," McGraw–Hill, New York, 1964.
13. S. ULAM, "A Collection of Mathematical Problems," pp. 69–70, Interscience, New York, 1960.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium

A Theory of Orbital Behavior in a Class of Nonlinear Systems: Chaos and a Signature-Based Approach

by

Martin E. Kaliski,
Northeastern University
Boston, MA *

S. Yunkap Kwankam
Universite de Yaounde
Yaounde, CAMEROUN

Pamela Halpern
Comp-All Systems, Inc.
Lynnfield, MA

and

David Shulman
Northeastern University
Boston, MA

* This author's work has been supported, in part, by the United States Air Force Office of Scientific Research, AFSC, Contract Number F49620-82-C-0080.

Mailing Address: Professor Martin E. Kaliski
405 Dana
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115

Abstract

A theory of orbital behavior in certain autonomous one-dimensional nonlinear systems is pursued, using an approach based upon the concept of (orbital) signature. Particular attention is paid to the fixed point structure of such systems with the ultimate aim of using the signature repertoires of these systems to characterize fixed-point orders and the presence of *chaos*. A system-theoretic approach is pursued here -- an approach which complements other recent studies of a more analytical nature. Chaotic behavior in a certain subclass of these system is completely characterized in terms of the first two iterates of a specific known point in the range of the system transition function.

Table of Notation

symbol	meaning
f	a generic unimodal or subbell function
f^k	for $k \geq 0$, the kth iterate of f
$sig(x)$	the (infinite) signature of x under f
$<$	ordinary order on the real line; Gray code order on binary strings
$sig^k(x)$	the k-signature of x under f
$ls(x), rs(x)$	the left,right signatures of x
$ls^k(x), rs^k(x)$	the left,right k-signatures of x
$/.../$	absolute value
S	the signature repertoire of f
S^k	the k-signature repertoire of f
$\cdot$	sequence concatenation
$\sim$	set intersection
$\cup$	set union
$s*j$	the jth rotation of s
Ω_k, A	the (k-)signature bins of a given string s
$L_j(s)$	the jth left shift of s
$[...]$	closed interval
$(...)$	open interval
$\subset$	subset of

I. Introduction

There has been great interest in recent years in the orbital behavior of autonomous nonlinear one-dimensional systems defined over the unit interval [0,1] [1,2,3,4,5,6,7]. There is, in particular, interest in the so-called *chaotic behavior* of the equilibrium, or *fixed*, points of such systems. This behavior manifests itself even in processes describable by simple first-order difference equations of the form:

$$x_{k+1} = f(x_k) \qquad (1)$$

where f: [0,1] -> [0,1].

Various approaches have been employed to analyze systems of form (1), ranging from graphical methods [8], to purely analytical techniques [4] and ergodic theoretic constructs [5]. A *system theoretic* approach, introduced by Klein and Kaliski (and cited above) is pursued in this paper. We view the function f as the state transition map of an autonomous one-dimensional nonlinear discrete-time system. The concept of *signature*, defined below, is used to characterize the orbit of any given initial system state.

This paper analyzes the fixed point structure of these systems through the use of *signature repertoires*. Our development is necessarily somewhat detailed due to our need to formalize and introduce certain fundamental concepts. Our results prove to be of intrinsic interest for the following two reasons:

(i) Chaotic behavior (the presence of fixed points of all periods) in a broad subclass of these systems -- the *signature-distinct well-structured unimodals* -- is characterized strictly in terms of the first two iterates of a specific point in the range of the system transition function. Hence a highly constructive test for chaos exists.

(ii) The utility of the signature concept is thus demonstrated, a concept with complements alternate methods that draw upon more advanced ergodic theory and measure-theoretic treatments of the subject.

Much of this work appeared in a somewhat different format in one of the authors' doctoral dissertation [9].

II. Basic Concepts: Unimodals, Subbells, Signatures and Gray Code Order

The concepts below were first introduced in the cited work of Klein and Kaliski. They are therefore just summarized below.

II.1 Unimodal and Subbell Functions

Definition 1: A *unimodal* function is a continuous map $f:[0,1] \to [0,1]$ for which (Figure 1):

(i) f has a unique maximum q at some point p in $(0,1)$

(ii) f is strictly monotone on $[0,p]$ and on $[p,1]$

We term p the *breakpoint* of f, and q its *peak value*.

Definition 2: A *subbell* function is a unimodal function obeying the additional constraint below (Figure 2):

(iii) $f(0) = f(1) = 0$

[The name *subbell* is derived from the relationship of these functions to the *bell* functions considered in reference 9; the bell function is additionally constrained to obey $f(p) = 1$]

This paper is primarily concerned, from an expositional point of view, with subbell functions, although many of the results rapidly generalize to the more extensive class of unimodal functions. In section VII we examine one key aspect of this issue in more depth -- the presence of chaotic regimes in unimodal functions. We see there how we can associate with a given unimodal an appropriate subbell and how orbital properties of the unimodal naturally derive from corresponding properties of the associated subbell function.

II.2 Finite and Infinite Signatures

Let f be a given subbell. Write, for $k > 1$, f^k to denote the kth iterate of f, i.e. the k-fold composition of f with itself. Define f^1 to be f and define f^0 to be the identity map, i.e. $f^0(y) = y$, for all y.

Definition 3: Let $k > 0$ be given, along with x in $[0,1]$. The *k-signature of x under f*, $\mathrm{sig}^k(x)$, is the length k string $b_0\, b_1 \ldots b_{k-1}$ where, for $i = 0,1,\ldots,k-1$:

$b_i = 0$, if $f^i(x)$ is in $[0,p)$

$= 1$, if $f^i(x)$ is in $(p,1]$

If $f^i(x) = p$, we set b_i to a "-".

$sig^k(x)$ will be called *regular* if, for all i, b_i is 0 or 1, but not "-". If $sig^k(x)$ is not regular, it will be called *irregular*.

The (infinite) *signature* of x, sig(x), is similarly defined as the infinite sequence of points in {0,1,-} obtained by letting k range over all the positive integers. We say that x is *regular* if sig(x) is regular; otherwise we term x irregular. Note that if x is regular then, for all k, $sig^k(x)$ is regular, and conversely.

II.3 Gray Code Ordering

We can define a total order upon binary strings (not necessarily regular signatures) which the reader will recognize as Gray code order:

Definition 4: Let $s^1 = b_0\, b_1 \ldots$ and $s^2 = d_0\, d_1 \ldots$ be two binary sequences of equal finite length, or of both infinite length. Then $s^1 < s^2$ if s^1 is not equal to s^2 and, denoting by j the bit position at which s^1 and s^2 first differ ($j \geq 0$),

$$(b_0 + \ldots + b_j) \bmod 2 = 0$$

$$(d_0 + \ldots + d_j) \bmod 2 = 1$$

This order relation is fundamental in the theory of signatures, in that all subbell functions obey a *monotonicity of signatures property* (monotone with respect to this order). We explore this in section II.5.

Note that we use the symbol "<" to denote both conventional numerical order on the reals as well as the just defined Gray code ordering on sequences. This will pose no problem in the sequel as the context of its use will always be apparent.

II.4 Instances of Irregular Signatures; Signature Repertoires

Definition 5: Suppose sig(x) is irregular for some x. An *instance* of sig(x) is any binary string obtained from sig(x) by arbitrarily inserting 0's and 1's for each "-" in sig(x). We term the least such instance obtained, in the < ordering, the *left-signature of x*, ls(x); we term the greatest such instance obtained the *right-signature of x*, rs(x). It is easy to show that ls(x) and rs(x) will differ in just one position, the position where the first "-" occurs. This is true even if sig(x) has more than one dash. When sig(x) is

irregular we define ls(x) = rs(x) = sig(x). In this case we can therefore refer to sig(x) as either ls(x) or rs(x) without ambiguity and will do so at times in the sequel, whenever convenient to.

These definitions extend in a straightforward way to finite signatures, and we write the left and right k-signatures of x as $ls^k(x)$ and $rs^k(x)$, respectively.

Definition 6: The *k-signature repertoire* of f, denoted by S^k, is the set of strings:

$$S^k = \{ rs^k(x),\ x \text{ in } [0,1] \} \cup \{ ls^k(x),\ x \text{ in } [0,1] \}$$

The *signature repertoire* of f, denoted by S, is the set of strings:

$$S = \{ rs(x),\ x \text{ in } [0,1] \} \cup \{ ls(x),\ x \text{ in } [0,1] \}$$

Thus the signature repertoires consist of all signatures of regular points, and the greatest and least signatures of all irregular points.

II.5 Monotonicity of Signatures

All subbell functions, as noted earlier, obey the following property. We cite it without proof. The proof appears in reference 2:

Theorem 1: Let f be a subbell; let x, y in [0,1] be given points, $x < y$. Then for all k, $rs^k(x) \leq ls^k(y)$, and $rs(x) \leq ls(y)$.

Note that in general we cannot strengthen the inequality "$\leq$" to "<". We return to this matter in section V.

III. A Roadmap for the Technical Developments to Follow

In Section IV we examine certain recursions for calculating the k-signature repertoires of subbells. We will see that the role of the left signature of the peak value q is central to these formulae. In Section V we begin to examine the issue of fixed point existence for subbells. Necessary and sufficient conditions for the existence of fixed points having given finite or infinite signatures are derived. We follow this in Section VI by an in depth examination of fixed point *orders* and the conditions necessary for the presence of *chaotic regimes* -- the presence of fixed points of all positive orders. Necessary background material is introduced as needed; the reader is often referred to the cited references for proofs of many of the subsidiary results presented.

Section VII examines the more general class of unimodal functions in the context of the earlier sections of the paper. We associate with an arbitrary unimodal a subbell for which our developed theory applies. This allows us to rapidly derive analogous properties for the unimodal family.

IV. Recursions for Determining Signature Repertoires

We present two recursions that tell us how to compute (k+1)-signatures from k-signatures.

Theorem 2: [2] For all $k \geq 1$

$$S^{k+1} = \{ 0 \cdot S^k \frown [\, ls^{k+1}(0),\ ls^{k+1}(p)\,] \} \cup$$

$$\{ 1 \cdot S^k \frown [\, rs^{k+1}(p),\ rs^{k+1}(1)\,] \}$$

where the $\cdot$ denotes sequence concatenation, $\frown$ denotes intersection, and the square brackets represent closed intervals in the $<$ ordering on binary sequences. We will often omit the concatenation symbol "$\cdot$" when its presence is implicit and unambiguous.

Since any subbell f obeys $f(0) = f(1) = 0$, it is immediate that $ls(0) = sig(0) = 000\ldots$ and $rs(1) = sig(1) = 100\ldots$ Further, ls(p), by definition, is equal to $0 \cdot ls(q)$; similarly $rs(p) = 1 \cdot ls(q)$. We may thus simplify the above recursion to read:

$$S^{k+1} = 0 \cdot \{ S^k \frown [\, 0^k,\ ls^k(q)\,] \} \cup$$

$$1 \cdot \{ S^k \frown [\, 0^k,\ ls^k(q)\,] \}$$

Observing that $S^1 = \{0,1\}$, we have the following alternative formulation of Theorem 2:

Theorem 3: The finite signature repertoires of a subbell are determined recursively as follows:

(i) $S^1 = \{0,1\}$

(ii) $S^{k+1} = 0 \cdot T^k \cup 1 \cdot T^k$,

where T^k consists of those k-signatures in S^k which are $\leq ls^k(q)$.

This formulation of the recursion underscores the significance of the peak value q of the subbell. In "signature space" subbells form a one parameter family of functions, the parameter being the left signature of the peak value of the subbell.

V. On the Existence of Fixed Points Having Given Finite and Infinite Signatures

We are concerned with the existence of fixed points of f^k, f a subbell, for various values of k, having given finite and infinite signatures. A fixed point of f^k is a point x_o for which $f^k(x_o) = x_o$ and represents a value which repeats with period k under iteration of f. To expedite the developments below we must introduce a few more notions.

V.1 On Rotations and Shifts of Sequences

Let s be a k-bit sequence. Denote by s*j the sequence obtained by rotating s circularly j bits to the left, $0 < j < k$. We term s*j the *jth left rotation* of s. Define s*0 to be equal to s.

Definition 7: The *rotational maximal* of s, rm(s), is the largest such left rotation of s. s is called *rotation maximal* if rm(s) = s.

The sequence 10011, for example, is rotation maximal. There may exist, in general, more than one value of j for which rm(s) = s*j. For example the sequence s = 011011 has rotational maximal rm(s) = 101101, which is equal to both s*2 and s*5. In all cases, however, rm(s) is well-defined and unique for a given s.

A similar concept applies for infinite binary sequences s. The jth *left shift* of s, $L_j(s)$, is the sequence obtained by shifting s j bits to the left, thus "lopping off" the j leftmost bits. Let L(s) denote the set of all left shifts of s, i.e. $L(s) = \{ L_j(s), s \geq 0 \}$. Define $L_o(s)$ to be s.

Definition 8: The *shift maximal* of s, sm(s), is equal to sup(L(s)). When s is equal to sm(s) we term s *shift maximal*.

The sequence 1001010111 ..., for example, is shift maximal. Note that in general sm(s) is not an element of L, as in the case when s = 10100100010001 ...; here sm(s) = 10000 ....

It can be proved [7] that ls(q) is always shift maximal, for any subbell. This is a useful fact that is used in the sequel.

V.2 Signature Bins

Let $s = b_o b_1 \ldots$ be a given infinite binary sequence.

Definition 9: The *k-signature bin* of s, $k \geq 1$, is the set

$$A_k = \{ x \text{ such that } ls^k(x) \text{ or } rs^k(x) = b_0\, b_1 \ldots b_{k-1} \}$$

From Theorem 1 (Monotonicity of Signatures) it follows that A_k, if non-empty, is an interval in [0,1]. In fact it is a closed interval [2].

Definition 10: The *signature bin* of s is the set

$$A = \{ x \text{ such that } ls(x) \text{ or } rs(x) = s \}$$

It again follows from Theorem 1 that A, if non-empty, is an interval in [0,1] and in fact satisifies [2]

$$A = \bigcap A_k$$

where the intersection is taken over all values of k. Hence A is a closed interval as well.

V.3 On Fixed Points of f^k and Their Signatures

We can now begin to turn to the central issues of this paper, the necessary background material in place. We will be addressing three questions in the pages that follow in this section:

QUESTION 1: When is an infinite binary sequence s in the signature repertoire S of a subbell f?

QUESTION 2: When is an infinite binary sequence s in S the left or right signature of a fixed point of f^k for some $k >= 1$?

QUESTION 3: When is a given length k binary sequence s the left k-signature or right k-signature of a fixed point of f^k for a given value of k?

We address these questions in turn through a sequence of Theorems and Lemmas. We begin with our first question:

Lemma 1: Let f be a subbell, s be a given infinite binary sequence. Then s is in S, the signature repertoire of f, if and only if, for all k, every k-truncation of s is in S^k, the k-signature repertoire of f. (By k-truncation we mean the first k bits of s).

Proof: The necessity of the condition is obvious, as the k-truncation of a left or right signature is a left or right k-signature. As for sufficiency, suppose every k-truncation for s is in S^k. Then each k-signature bin A_k is non-empty, and as noted, is closed. Thus $A = \bigcap A_k$ is non-empty also. But by our earlier remarks, A is the signature bin of s. Thus s is in S. QED

We can now provide a complete answer to Question 1:

Theorem 4: Let s be a given infinite binary sequence. Then s is in S, the signature repertoire of f, if and only if

$$L_j(s) \leq ls(q), \text{ for all } j > 0$$

Proof: (->) Suppose s is in S and is equal to ls(x) or rs(x) for some point x in [0,1]. Every shift of s is in S and is the left-signature or right-signature of a point in the orbit of x. Such points, from the definition of q as the peak value of f, have values less than or equal to q. If q is not in the orbit of x then, by Theorem 1, $L_j(s) \leq ls(q)$ for all $j > 0$, and we are done. If q is in the orbit of x then so is p, the breakpoint of f, and it is easy to verify that all shifts of s corresponding to those iterates of x equal to q are equal to ls(q). All others, by Theorem 1, are less than or equal to ls(q). The conclusion follows.

(<-) Suppose that $L_j(s) \leq ls(q)$ for all $j > 0$. Write s as $b_0\ b_1\ \ldots$ We are going to use the recursion of Theorem 3. Since b_0 is clearly either 0 or 1, it is in S^1, as is b_1. Since b_1 is the first bit of $L_1(s)$, and $L_1(s) \leq ls(q)$, by hypothesis, we have $b_1 \leq ls^1(q)$ and thus deduce by Theorem 3 that $b_0\ b_1$ is in S^2. Now consider $b_0\ b_1\ b_2\ \ldots$ It is easy to deduce by an argument similar to that just given that $b_1\ b_2$ is in S^2; further, $b_1\ b_2$ are the first two bits of $L_2(s)$. Thus, with $L_2(s) \leq ls(q)$, again by hypothesis, $b_1\ b_2 \leq ls^2(q)$. So again by Theorem 3, $b_0\ b_1\ b_2$ is in S^3. By repeated use of these arguments, we find that every k-truncation of s is in S^k. From Lemma 1, then, s is in S, as desired. QED

Having dispensed with our first question, we can now turn to Question 2. We must settle for a partial answer, however, for although it is certainly true that the signature of a fixed point of f^k is periodic with period k, its left or right signature may not be (one always will be as we will see shortly.)

Theorem 5: Let $k \geq 1$ be given, along with a subbell f, and infinite binary sequence s in S. Then if s is periodic with period k there is a fixed point x of f^k for which ls(x) or rs(x) = s.

Proof: Let A_i denote the i-signature bin of s, and A the signature bin of s. Since s is in S, all the A_i and A are non-empty closed intervals in [0,1]. Now it is certainly the case that $A_1 \supset A_2 \supset \ldots$, and since $A = \sim A_i$, it is also true that $A = \sim A_{ik}$, $i = 1,2,\ldots$

From the assumed periodicity of s we know that $f^k(A_{ik}) \subset A_{(i-1)k}$ for all $i \geq 2$. Thus $\sim f^k(A_{ik}) \subset \sim A_{(i-1)k}$, where the first intersection begins with i=1, the second with i=2. But we may re-write the latter as $\sim A_{ik}$, with i beginning at 1, and this equals A, by our observation in the preceding paragraph. Thus $\sim f^k(A_{ik}) \subset A$. It is easy to deduce that $f^k(\sim A_{ik}) \subset \sim f^k(A_{ik})$. Thus $f^k(A) = f^k(\sim A_{ik}) \subset A$, and thus f^k has a fixed point x in A. By definition of A, then, ls(x) or rs(x) equals s. QED

Corollary: A given infinite binary sequence s is the left or right signature of a fixed point of f^k if it is periodic of period k, and if $sm(s) \leq ls(q)$

Proof: Periodic infinite sequences always contain their shift maximals. Thus all left shifts of s will in fact be less than or equal to ls(q). From Theorem 4, then, s will be in S. The result follows from Theorem 5. QED

Before addressing the third of our questions we need to state the finite sequence analog of Theorem 4:

Theorem 6: Let s be a k-bit sequence for which $rm(s) \leq ls^k(q)$. Then every rotation of s is in S^k.

Proof: We will show that every rotation s_i of s satisfies the following properties: if s_i begins with a 0, $s_i \leq ls^k(p)$; if s_i begins with a 1, then $s_i \geq rs^k(p)$. By a basic result of reference 9 it will follow that every rotation of s is in S^k. (Lack of space prohibits repeating the proof of this result.)

Assume, then, that there is some rotation s_i of s for which $s_i = 0\ b_1\ b_2 \ldots > ls^k(p)$ or $s_i = 1\ b_1\ b_2 \ldots > rs^k(p)$. Since $ls^k(p) = 0\ ls^{k-1}(q)$ and $rs^k(p) = 1\ ls^{k-1}(q)$, we have in both cases that $b_1\ b_2 \ldots > ls^{k-1}(q)$. Now for any binary sequences z_1 and z_2 of equal length, and z_3 and z_4 of equal length, $z_1 > z_2$ implies $z_1\ z_3 > z_2\ z_4$. Setting $z_1 = b_1\ b_2 \ldots$, $z_2 = ls^{k-1}(q)$, $z_3 = 0$ or 1 (according to the case) and z_4 = the kth bit of ls(q) yields $s_i * 1 > ls^k(q)$.

But $s_i * 1 \leq rm(s)$, which by hypothesis is $\leq ls^k(q)$. Thus we arrive at a contradiction and no such rotation s_i of s exists. The proof is complete. QED

Our next result is an important one, both for addressing Question 3, and for the developments to follow in our analysis of chaotic behavior.

Theorem 7: Let x_o be a fixed point of f^k. Then either $ls(x_o)$ or $rs(x_o)$ is periodic of period k. Let s denote the first k-bits of this periodic signature and term it the *periodic k-signature of* x_o. There exists x_1 in the orbit of x_o whose left k-signature is equal to rm(s). Further, rm(s) rm(s) $\leq$ $ls^{2k}(q)$ and, when it is equal to $ls^{2k}(q)$, ls(q) is in fact periodic and equal to rm(s) rm(s) rm(s) ...

Proof: Clearly $sig(x_o)$ is periodic with period k. If $sig(x_o)$ is regular then s is equal to $sig^k(x_o)$ and it is readily apparent that rm(s) occurs as a subsequence of $sig(x_o)$. If this subsequence begins for the first time at position $j \geq 0$ in $sig(x_o)$, then rm(s) is the k-signature of $x_1 = f^j(x_o)$. Further, $sig(x_1)$ = rm(s) rm(s) ... Note that since p, and hence q, does not occur in the orbit of x_o all points in this orbit (including x_o) are less than q. From the Monotonicity of Signatures Property, then, rm(s) rm(s) $\leq$ $ls^{2k}(q)$. If it is $<$ $ls^{2k}(q)$ there is nothing further to prove in the $sig(x_o)$ regular case; assume that it equals $ls^{2k}(q)$, then.

Write rm(s) as s_1. Thus $sig(x_1) = s_1\ s_1 \ldots$ Again, using Monotonicity of Signatures, $sig(x_1) \leq ls(q)$. If it is less than ls(q) we argue as follows. Its first 2k bits agree with ls(q) by hypothesis. It must be then that ls(q) is of the form $s_1\ s_1 \ldots s_1$ (n times) z_1, where $n \geq 2$, and where the infinite sequence z_1 does not begin with s_1. Now suppose $sig(x_1)$ and ls(q) differ for the first time at the nk+j th bit, with $0 < j < k$. Let m = nk if n is even, and (n-1)k if n is odd. Consider $L_m(sig(x_1))$ and $L_m(ls(q))$. The former is again $sig(x_1)$, whereas the latter is z_1 if n is even and $s_1\ z_1$ if n is odd. In both cases (n odd, n even), an even number of s_1's were deleted and hence $sig(x_1) < L_m(ls(q))$. Since these two sequences differ within the first 2k bits (as z_1 does not begin with s_1), it must be that the first 2k bits of $L_m(ls(q))$ are greater than those of $sig(x_1) = ls^{2k}(q)$. But this is not possible by Theorem 4; Thus $sig(x_1) = ls(q)$ and the latter is periodic and equal to rm(s) rm(s) ...

There just remains the $sig(x_o)$ irregular case to discuss. Clearly both p and q occur in the orbit of x_o, and because $sig(x_o)$ is periodic no point greater than q occurs in x_o's orbit. It is straightforward to demonstrate that ls(q) will be periodic of period k and that either $rs(x_o)$ or $ls(x_o)$ will be, depending upon the parity of $sig(x_o)$ before the first "-". After this "-" the remainder of both $ls(x_o)$ and $rs(x_o)$ is ls(q). Since s is the first k-bits of this signature it follows that rm(s) must be $ls^k(q)$ by monotonicity of signatures. The point x_1 having rm(s) as its left-signature is q itself, and thus $ls(x_1)$ is equal to ls(q), which is thus rm(s) rm(s) ... Note, in particular, that rm(s) rm(s) is equal to $ls^{2k}(q)$. QED

We use Theorem 7 to answer Question 3. In fact, we couch the answer as a Corollary to Theorem 7:

Corollary: Let s be a given k-bit binary sequence. Then there is a fixed point of f^k with periodic k-signature s if and only if rm(s) · rm(s) $\leq$ $ls^{2^k}(q)$, with ls(q) = rm(s) rm(s) ... when rm(s) rm(s) = $ls^{2^k}(q)$.

Proof: (->) From Theorem 7

(<-) Conversely, consider the sequence z = s s s ... (n times), for any n > 1. It is easy to demonstrate that rm(z) = rm(s) rm(s) ... (n times). It follows from our hypothesis that rm(z) $\leq$ $ls^{n^k}(q)$. Thus from Theorem 6, z itself is in S^{n^k}. Hence all initial portions of z of length m, m = 1,...,nk-1 are in S^m. Since this is true for all n > 1 it follows that s s s ... is in S, by Lemma 1. Since the latter infinite sequence is periodic with period k, we know by Theorem 5 that there is a fixed point x of f^k whose left or right signature is equal to s s s .... Certainly s is x's periodic k-signature. QED

V.4 Signature-Distinct Subbells

Definition 11: A subbell is termed *signature-distinct* if sig(x) = sig(y) implies x = y. A wide class of signature-distinct subbells have been shown to exist [2] and include all those subbells which are *piecewise strictly expansive*, i.e. for which there exists E > 1 such that for all x,y in [0,p], /f(x)-f(y)/ $\geq$ E /x-y/ and similarly for all x,y in [p,1]. One consequence of signature-distinctness is that if x is not equal to y, then no instance of sig(x) is equal to an instance of sig(y) either.

For signature-distinct subbells a more powerful form of the Corollary to Theorem 7 holds:

Theorem 8: Let f be signature-distinct and let s be a given k-bit binary sequence. Then there is a fixed point of f^k with periodic k-signature s if and only if rm(s) rm(s) $\leq$ $ls^{2^k}(q)$, with the further proviso, in the case of equality, that q is a fixed point of f^k as well.

Proof: (->) If such a fixed point exists then rm(s) rm(s) $\leq$ $ls^{2^k}(q)$ from the above Corollary to Theorem 7. Furthermore, in the case of equality, ls(q) is periodic with period k and is equal to rm(s) rm(s) · ... By Theorem 5, then, there exists a fixed point x of f^k whose left or right signature is equal to ls(q). Since f is signature-distinct this point x must in fact equal q.

(<-) Conversely, if rm(s) rm(s) < $ls^{2^k}(q)$ the result is immediate from the Corollary to Theorem 7. If equality holds and q is a fixed point of f^k as well, then it is easy

to see that, first of all, ls(q) is periodic with period k, and that, secondly, then, ls(q) must in fact equal rm(s) rm(s) ... Again using the Corollary to Theorem 7, the result follows. QED

VI. On Fixed Points and Their Orders

In this section we turn to characterizing the *fixed point regimes* of subbell functions, building upon the material of section V. Let f be a given subbell in the discussion that follows.

Definition 12: A finite binary sequence s of length $n \geq 1$ is said to be of *order* k, $k \leq n$ if we can write s in the form $s_1\ s_1 \ldots s_1$ (p-times), where s_1 is of length k, $n = pk$, and s_1 cannot be similarly decomposed.

(Thus a sequence consisting of only 1's or only 0's is of order 1; the five bit sequence 10110 is of order five, and the six bit sequence 010010 is of order three.)

Definition 13: x_o is a *fixed point of f of order k*, $k \geq 1$, if

(i) $f^k(x_o) = x_o$

(ii) $f^j(x_o)$ is different from x_o, for $j = 1,\ldots,k-1$

Thus a fixed point of order k first maps into itself after k iterations under f. Note that fixed points of order 1 are simply defined as those points that obey (i) for $k = 1$. Note also that if x is a fixed point of f^k and its "periodic" k-signature is of order k, then x is in fact of order k. We state this as a Lemma:

Lemma 2: If x is a fixed point of f^k and its "periodic" k-signature is of order k, then x is of order k.

Proof: By contradiction. Suppose x is of order $n < k$. Clearly n divides k, and x is also a fixed point of f^n. We know from Theorem 7 that either ls(x) or rs(x) is periodic of period n. Suppose it is ls(x). Then ls(x) is also periodic of period k, since n divides k. Thus the "periodic" k-signature of x consists in this case of k/n copies of the "periodic" n-signature of x. A similar remark holds if it is rs(x) that it is periodic. In both case then the order of the "periodic" k-signature of x is at most n. QED

Every subbell with peak value q greater than its breakpoint p has a (regular) fixed point of order 1, having signature 111 ... Term this point the *non-trivial* fixed point of order 1 of f. (f has a *trivial* fixed point at 0, of course.) For all k, the k-signature of f's non-trivial fixed point is 1^k, and, by definition, is in S^k. Since the

condition $q > p$ can be equivalently phrased as $ls^1(q) = 1$, we can summarize this simple observation as

Lemma 3: If $ls^1(q) = 1$, then, for all $k \geq 1$, the sequence 1^k is in S^k.

We are motivated to next look at certain k-bit sequences containing a single 0. The reasons for this will be made clear below. By imposing a somewhat stronger condition upon q we have:

Theorem 9: If $ls^2(q) = 10$ then, for all $k \geq 2$, 101^{k-2} is in S^k.

Proof: Clearly the Theorem holds for $k = 2$. Assume then that $k > 2$. Now the conditions of this Theorem are stronger than those of the above Lemma; hence 1^{k-2} is in S^{k-2}. From Theorem 3, we then have that 01^{k-2} is in S^{k-1}, since it is clear that $1^{k-2} < ls^{k-2}(q)$ as ls(q) begins with 10. Similarly 101^{k-2} is in S^k, since $01^{k-2} < ls^{k-1}(q)$ also. QED.

The condition $ls^2(q) = 10$ is a fundamental one for subbells, since it must hold if any fixed points of order greater than two are present. Since it is the study of such fixed points that is of particular interest here, we will restrict our attention to such subbells, terming them *well-structured*. We of course need to demonstrate the above claim.

Theorem 10: If $ls^2(q)$ is not equal to 10, then f has no fixed points of order greater than two.

Proof: We essentially argue by contradiction, looking at the various ways that $ls^2(q)$ can not equal 10.

Case 1: ls(q) begins with a 0

The argument: In this case, $q \leq p$. Since q is the peak value of f, it follows that for all x in [0,1], $f(x) \leq p$. Assuming that x is a fixed point of order greater than two it must be that $f(x)$ and $f^2(x)$ are unequal. If $f^2(x)$ is less than $f(x)$, then by the monotonicity of f on [0,p] and the above observation that range(f) is a subset of [0,p], it is immediate that for all $j \geq 1$, $f^{j+1}(x) < f^j(x)$. In other words, $\{f^j(x)\}$ is a decreasing sequence and thus x is not a fixed point. Contradiction.

One can similarly argue the case where $f^2(x)$ is greater than $f(x)$, deducing that $\{f^j(x)\}$ is increasing, and that x is thus not a fixed point. Again a contradiction arises.

Case 2: ls(q) begins with 11

The argument: Suppose x is a fixed point of order greater than two. There are several subcases based upon the form that the signature of x may take.

case 2a: sig(x) contains no 1's

No iterates of x are greater than p. In particular it must be that $x < f(x) \leq p$ or $f(x) < x \leq p$. (x cannot equal f(x), or it would be a fixed point of order 1). We can argue as in Case 1 that $\{f^j(x)\}$ is either an increasing or decreasing sequence, and hence x cannot be a fixed point.

case 2b: sig(x) contains no 0's

No iterates of x are less than p. Let us say, for real numbers a,b,and c in [0,1] that b is *between* a and c if either $a < b < c$ or $c < b < a$. It is easy to prove that if a,b, and c are all greater than or equal to p, then if b is between a and c, f(b) is between f(a) and f(c).

With the order of x greater than two, x, f(x), and $f^2(x)$ are all different. Thus one of them is between the other two. We thus have three possibilities for "case 2b".

1) If f(x) is between x and $f^2(x)$ we reason as follows: if $f(x) > x$ then from the definition of "between", on the one hand, $f(x) < f^2(x)$, but with all iterates of $x \geq p$, on the other hand, $f(x) > f^2(x)$. Contradiction. A similar contradiction arises if $f(x) < x$. Thus f(x) is not between x and $f^2(x)$.

2) If $f^2(x)$ is between x and f(x) we argue as follows: By our observation above, for all $i \geq 0$, $f^{i+2}(x)$ is between $f^i(x)$ and $f^{i+1}(x)$. But this implies that $|f^{i+2}(x)-f^{i+1}(x)| < |f^{i+1}(x)-f^i(x)|$, and hence the distance between successive iterates of x is strictly decreasing. But this sequence of distances must be periodic if x is a fixed point. Contradiction.

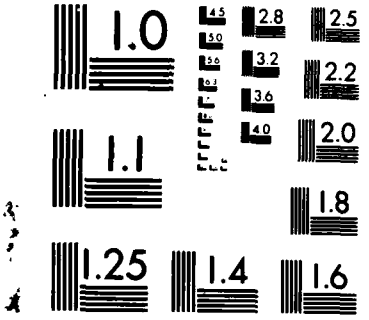
3) If x is between f(x) and $f^2(x)$ the argument is similar to that of possibility "2" above. For all $i > 0$, $f^i(x)$ is between $f^{i+1}(x)$ and $f^{i+2}(x)$, and thus the distance between successive iterates of x is strictly increasing, again leading to a contradiction.

case 2c: sig(x) contains both 0's and 1's

This is the last and most interesting case to consider. We have noted that ls(q) is shift maximal; hence, since it begins with 11, it must in fact be equal to 11... Now with x a fixed point all its iterates must be $\leq q$, since they are obviously in the range of f. Hence, by Theorem 4, the left and right signatures of all of x's iterates must be

AD-A174 525 ASYNCHRONOUS DISCRETE CONTROL OF CONTINUOUS PROCESSES (U) NORTHEASTERN UNIV BOSTON MA M E KALISKI 24 FEB 86 AFOSR-TR-86-2052 F49620-82-C-0080 3/3

UNCLASSIFIED F/G 9/4 NL



$ls(q) = 111...$ Thus no such left or right signature can be of the form 10... It is easy to argue from this that with sig(x) periodic of period greater than 2, and of necessity containing 0's and 1's, this constraint cannot be met.

This completes the proof of the Theorem. QED

Theorem 11: Let f be well-structured. Then f has fixed points of order k, k odd, and greater than or equal to 3, if and only if it has a fixed point x_0 of f^k with "periodic" k-signature equal to 101^{k-2}.

Proof: (<-) Since the sequence 101^{k-2} is of order k, it follows that x_0 is of order k from Lemma 2.

(->) Let y_0 be a fixed point of order k, k odd, $k > 3$. Let s be its "periodic" k-signature, and assume s is not equal to 101^{k-2}. s cannot equal 0^k, for if it did, then either $ls(y_0)$ or $rs(y_0)$ would be 000... (in fact it would be $ls(y_0)$) and thus y_0 and all its iterates would be $\leq$ p. By arguing as for "Case 1" in the proof of Theorem 10 we arrive at the conclusion that y_0 cannot be a fixed point of order greater than 2, a blatant contradiction. Thus s is not 0^k. Similarly s cannot be 1^k, for we would have $ls(y_0)$ or $rs(y_0) = 111...$ and thus y_0 and all of its iterates would be $\geq$ p. By arguing as in "case 2b" in the proof of Theorem 10 we conclude again that y_0 is not a fixed point.

Thus s is not 0^k or 1^k. It it is easy to prove[7] that if k is odd and s is different from 0^k or 1^k, then $rm(s) \geq 101^{k-2}$. From Theorem 7, $rm(s)\ rm(s) \leq ls^{2k}(q)$. Thus we have $101^{k-2}101^{k-2} \leq ls^{2k}(q)$ as well. Using Theorem 7 again, the proof is complete. QED

We next prove a result which shows how the presence of fixed points of certain orders implies the existence of fixed points of other orders.

Theorem 12: Let f be well-structured. If f has a fixed point of order k, k odd, $k > 1$, then f has fixed points of orders k-1,k+1, and k+2.

Proof: Assume f has a fixed point of order k, k odd and greater than 1. From Theorem 11 there is a fixed point x_0 of f^k whose periodic k-signature is 101^{k-2}; from the Corollary to Theorem 7, using the rotation maximality of 101^{k-2}, we thus have $101^{k-2}101^{k-2} \leq ls^{2k}(q)$.

Consider the sequence 101^{k-1}, which we will denote by s_1. It is of order k+1 and is itself rotation maximal. If we can argue that $s_1s_1 < ls^{2k+2}(q)$ it will follow from this Corollary, that there is a fixed point of f^{k+1} with periodic (k+1)-signature s_1. From Lemma 2 it will follow that this fixed point is of order k+1.

We do argue this as follows. Look at the leftmost k+2 bits of s_1s_1 and of $101^{k-2}101^{k-2}$. The former is 101^k; the latter $101^{k-1}0$. Since k is odd, 101^k is less than $101^{k-1}0$. Thus the leftmost 2k bits of s_1s_1 are less than $101^{k-2}101^{k-2}$ and thus less than $ls^{2k}(q)$; so s_1s_1 itself is less than $ls^{2k+2}(q)$, as desired.

The remainder of the proof is along similar lines. To obtain the fixed point of order k+2 we look at the sequence $s_2 = 101^k$. It is rotational maximal, of order k+2 and can readily be shown to be obey $s_2s_2 < ls^{2k+4}(q)$. From the Corollary to Theorem 7, and from Lemma 2, we deduce the existence of a fixed point of f^{k+2} which is of order k+2.

To obtain the fixed point of order k-1 we look at the sequence $s_3 = 101^{k-3}$. This is of order k-1, is rotational maximal and obeys $s_3s_3 < ls^{2k-2}(q)$. The proof is complete. QED

Thus when a well-structured subbell has fixed points of order k, k odd and > 1, it has fixed points of orders k-1,k+1, and k+2. Since k+2 is odd, we can re-apply Theorem 12 to deduce the presence of fixed points of orders k+3, and k+4, and, continuing in this fashion we come to the conclusion that *if f has fixed points of order k_o, k_o odd and > 1, it has fixed points of all orders $\geq k_o-1$.*

In particular, then, it follows that if a well-structured subbell has fixed points of order three it has fixed points of all orders, since it certainly has the non-trivial fixed point of order 1. This conclusion is akin to that derived in reference 4; unlike the results derived therein, however, our "test" for this *chaotic* condition is far simpler in the case that f is signature-distinct:

Theorem 13: Let f be an *arbitrary* well-structured signature-distinct subbell. Then f has fixed points of all orders if and only if

$$rs^3(q) = 100$$

Proof: (->) Suppose the above condition on $rs^3(q)$ does not hold. We will show that f has no fixed point of order three, contradicting the hypothesis of the Theorem. The sequence 100 is the largest three bit sequence. If $rs^3(q)$ *does not equal* 100, then, it must be less than it. Combining this observation with $ls^2(q) = 10$, it is immediate that $sig^2(q) = 10$ and $rs^3(q) = ls^3(q) = 101$. Thus $sig^3(q)$ is regular, and q is not a fixed point of f^3.

The sequence ls(q) is shift-maximal; since $ls^3(q) = 101$, we thus have $ls^6(q) \leq 101101$. We thus deduce that f has no fixed point with periodic 3-signature 101, for were there to be one, it would follow from the Corollary to Theorem 7, that $ls^6(q) = 101101$, and that, further, by Theorem 8, that q is a fixed point of f^3, contradicting the above remarks. But if f has no fixed point with periodic 3-

signature 101 it has no fixed points of order 3 at all from Theorem 11. The conclusion: $rs^3(q) = 100$.

(<-) Suppose that $rs^3(q) = 100$. If $sig^3(q)$ is regular, then $ls^3(q)$ is also 100, and thus $101101 < ls^6(q)$. By Theorem 7's Corollary, then, f^3 has a fixed point with three bit periodic signature 101. This is clearly a fixed point of order 3, and thus by our remarks preceding the statement of this Theorem, f has fixed points of all orders. If $sig^3(q)$ is irregular then, since $ls^2(q) = 10$ (f is well-structured) it must be that $sig^3(q) = 10-$. Thus q is a fixed point of f^3, since p maps to q. Further, its order is 3. It then follows again that f has fixed points of all orders. QED

We therefore need examine only the first three bits of the right signature of the peak value q of a well-structured signature-distinct subbell to determine the presence of chaos.

From the viewpoint of economy of computation, we need only look at how p compares with f(p), $f^2(p)$, and $f^3(p)$. As an example, consider the family of "symmetric tent" maps shown in Figure 3. These maps are all signature-distinct since with $a > 1/2$ they are piecewise strictly expansive. Further, when $a > 1/2$ it is easy to deduce that $ls^2(q) = 10$; they are thus well-structured. It is readily demonstrated that $rs^3(q)$ will equal 100 if and only if $a \geq (1+\sqrt{5})/4$, i.e. chaos will be present when a is in $[(1+\sqrt{5})/4, 1]$.

VII. Unimodals and Subbells. Extensions of the Theory.

Let us turn, in this final section, to the case of the more general unimodal function (Definition 1). Although we could directly re-examine, result by result, our just developed theory to see which results generalize to unimodal functions, we pursue a more interesting approach in the paragraphs that follow. For lack of space we focus on generalizing just the key result of this paper -- Theorem 13.

Let f be a given unimodal, with $f(0) = a$, $f(1) = b$, $0 < a,b < 1$, and breakpoint p in (0,1), peak value $q = f(p)$. Define the auxiliary map $h: [0,1] \rightarrow [1/4,3/4]$ via $h(x) = (2x+1)/4$. Thus h is a linear map, mapping 0 to 1/4 and 1 to 3/4. The map $h^{-1} : [1/4,3/4] \rightarrow [0,1]$ is, of course, defined as well, and obeys $h^{-1}(y) = (4y-1)/2$. The reader will observe that there is nothing "sacred" in the values 1/4 and 3/4; we have chosen them with the goal of appending simple "steep linear legs" to the unimodal -- a fact that will be apparent from the discussion below. The reader will also observe that we have assumed that *both* a and b are positive. If either is, in fact, 0, then a straightforward modification of the arguments below will be called for.

These cases entail creating a "single-legged" subbell. If both a and b are 0 we have the case of the subbell itself, of course, and there is nothing to discuss.

Consider the mapping f*: [0,1] -> [0,1] defined as follows:

$$f^*(z) = 4h(a)z, \text{ if } 0 \leq z \leq 1/4$$

$$= hfh^{-1}(z), \text{ if } 1/4 \leq z \leq 3/4$$

$$= 4h(b)(1-z), \text{ if } 3/4 \leq z \leq 1$$

These various maps are all shown in Figure 4. Note, in particular that f* maps [1/4,3/4] into [1/4,3/4]. Thus any point that gets mapped by f* into [1/4,3/4] becomes "trapped" there under further iteration. This concept is central to the results prsented below.

Lemma 4: f* is a subbell with breakpoint p* = h(p), and peak value q* = h(q).

Proof: Note first that f* is continuous and that f*(0) = f*(1) = 0. Further, consider the behavior of f* on [0,h(p)]. h(p) of necessity is > 1/4, since p is > 0; it is also < 3/4, since p < 1. Thus only the first two equations above apply. For z in [0,1/4], of course, f*(z) is strictly monotone increasing; for z in [1/4,h(p)], $h^{-1}(z)$ strictly increases from 0 to p; $fh^{-1}(z)$ thus strictly increases from a to q, and thus f*(z) strictly increases from h(a) to h(q). Conclusion: f* is strictly monotone increasing on [0,h(p)]. By similar reasoning f* is strictly monotone decreasing on [h(p),1]. The Lemma follows. QED

Term this artificially created subbell the *associated subbell* of the unimodal f. f* obeys several more interesting properties. We begin by noting that the concepts of fixed points and their orders, although defined for subbells, naturally extend to unimodals as well.

Lemma 5: Suppose that x in (0,1) is a fixed point of f^{*k}, for some k > 0. Then x in fact must be in [1/4,3/4] and, further, $h^{-1}(x)$ is a fixed point of f^k. If x is of order k, then so is $h^{-1}(x)$.

Proof: We first argue that x must be in [1/4,3/4]. If not, there are two possibilities: x is in (0,1/4), or x is in (3/4,1). In the former case the iterates of f* on x increase as 4h(a)x, $(4h(a))^2x$,... until one of these values leaves the interval (0,1/4). This will happen, since, with a in (0,1), 4h(a) is in (1,3), and, in particular, is greater than 1. When it leaves (0,1/4) it must thus go to a value in [1/4,3/4]. It will then be captured in [1/4,3/4], as noted earlier. Thus x can never return to (0,1/4) and

is not a fixed point of f^{*k}, for any value of $k > 0$. A similar, but more subtle, argument allows us to conclude that x is not in (3/4,1).

Points in (3/4,1) of necessity map into (0,h(b)). If x is in (1-1/16h(b),1) it maps to (0,1/4), and by the argument above, can never return under subsequent iterations to (3/4,1). If x is in (3/4,1-1/16h(b)) it maps to (1/4,h(b)); but this is a subset of (1/4,3/4) and hence x again is captured. Conclusion: if such a fixed point of f^{*k} exists it must be in [1/4,3/4].

The remainder of the proof comes quickly. With [1/4,3/4] closed under f^* we argue as follows: $f^{*k}(x) = hf^kh^{-1}(x)$ with x in [1/4,3/4]. Thus if $f^{*k}(x) = x$, it follows immediately that $f^k(h^{-1}(x)) = h^{-1}(x)$. That the orders of x and $h^{-1}(x)$ agree is clear. QED

Corollary: Other than the trivial fixed point of 0, all other fixed points of f^* are in [1/4,3/4] and are images under h of fixed points of f of the same order.

The "converse" to this Lemma also holds and we state it without proof:

Lemma 6: If x is a fixed point of f^k, for some $k > 0$, then h(x) is a fixed point of $f^*(x)$. This fixed point is in [1/4,3/4] and its order agrees with that of x.

Corollary: f has fixed points of all orders if and only if f^* does, i.e. f is chaotic if and only if f^* is.

Proof: Necessity is immediate from Lemma 6. As for sufficiency we note that all fixed points of f^* of order greater than 1 are in [1/4,3/4] by Lemma 5, and are images of fixed points of f of the same order. f^* has a fixed point of order one in [1/4,3/4] also; thus f has a fixed point of order one. QED

Finally we explore the issue of chaos. The notions of signatures, well-structuredness, and signature-distinctness generalize rapidly to unimodals in a straightforward way.

Theorem 14: If f is well-structured and signature-distinct then so is f^*. If f is so behaved then f has fixed points of all orders if and only if $rs^2(q) = 100$, where the signature is taken with respect to f.

Proof: We begin by noting that for any x in [0,1] the f-signature of x is identical with the f^* - signature of h(x). Similarly, for x in [1/4,3/4], the f-signature of $h^{-1}(x)$ is identical with the f^*-signature of x. Thus if $ls^2(q) = 10$ for f it must be that $ls^2(q^*) = 10$ for f^*, and conversely. In other words f is well-structured if and only if f^* is. The signature-distinctness of f certainly implies the

signature-distinctness of f^* over the interval [1/4,3/4].

That f^* is signature-distinct over *all* of [0,1] results from the following argument.

Define the *capture time* of a point x in [0,1] to be the smallest $k \geq 0$ for which $f^{*k}(x)$ is in [1/4,3/4]. Clearly if x itself is in [1/4,3/4] its capture time is equal to 0. Take two arbitary distinct points x_1 and x_2 in [0,1]. Let k_1 and k_2 denote their respective capture times. Set $k = \max(k_1,k_2)$. One of two cases occurs. $sig(x_1)$ and $sig(x_2)$ either agree through the first k+1 positions, or they do not. In the latter case the signatures differ, obviously, and we are done. In the former case it follows that $y_1 = f^{*k}(x_1)$ and $y_2 = f^{*k}(x_2)$ are also distinct (by the strict piecewise monotonicity of f^* and the obvious fact that we have faithfully tracked pieces together to this point). Furthermore, all their iterates are in [1/4,3/4], by the "capturing property". By the just deduced signature-distictness of f^* on [1/4,3/4] then, $sig(y_1)$ will differ *from* $sig(y_2)$.

Thus if f is signature-distinct, f^* is. The remainder of this Theorem follows immediately from the fact that if $rs^3(q) = 100$ for f, then $rs^3(q^*) = 100$ for f^*. Applying Theorem 13 and the above Corollary completes the proof. QED

VIII. Conclusions

This paper has presented a variety of results concerning the fixed point structure of certain maps defined over the unit interval. The underlying common thread of the developed theory has been that of the signature of a point and of its role in characterizing the map's orbital behavior.

From an expositional point-of-view this signature-based theory is appealing; it is minimally dependent upon advanced measure-theoretic concepts typically found in the literature, and needs no a priori assumptions on functional form such as differentiability, linearity, or convexity.

The authors are currently addressing the most significant restriction inherent in parts of this paper: signature-distinctness. We are, more specifically, seeking to see whether Theorems 13 and 14 indeed hold when the signature-distinctness condition is removed, or whether this condition is indeed necessary. Appropriate counterexamples will be produced in this latter case.

References

1. Kaliski, M. and Klein, Q., "Behavior of a Class of Nonlinear Discrete- Time Systems," *Journal of Computer and System Sciences*,31,1,1985.

2. Klein, Q. and Kaliski, M., "Functional Equivalence in a Class of Autonomous One-Dimensional Nonlinear Discrete-Time Systems," *Information and Control*,42,2,1979.

3. Collett,P. and Eckmann, J., *Iterated Maps on the Interval as Dynamical Systems*, Birkhauser, 1980.

4. Li, T. and Yorke, J., "Period Three Implies Chaos," *American Mathematical Monthly*,82,1975.

5. Parry, W., "Symbolic Dynamics and Transformations of the Unit Interval," *Transactions of the American Mathematical Society*,122,1966.

6. Baillieul,J., Brockett,R., and Washburn, R., "Chaotic Motion in Nonlinear Feedback Systems," *IEEE Transactions of Circuits and Systems*,CAS-27,11,1980.

7. Feigenbaum,M., "Universal Behavior in Nonlinear Systems," *Los Alamos Science*, 1980.

8. May,R. and Oster,G., "Bifurcations and Dynamic Complexity in Simple Ecological Models," *American Naturalist*,1976.

9. Kwankam, S., "A Structure Theory for the Fixed Points of the Iterates of Certain One-Dimensional Nonlinear Functions Defined Over the Unit Interval," Ph.D. Dissertation, Northeastern University Department of Electrical Engineering, Boston, MA, 1979.

LIST OF FIGURES

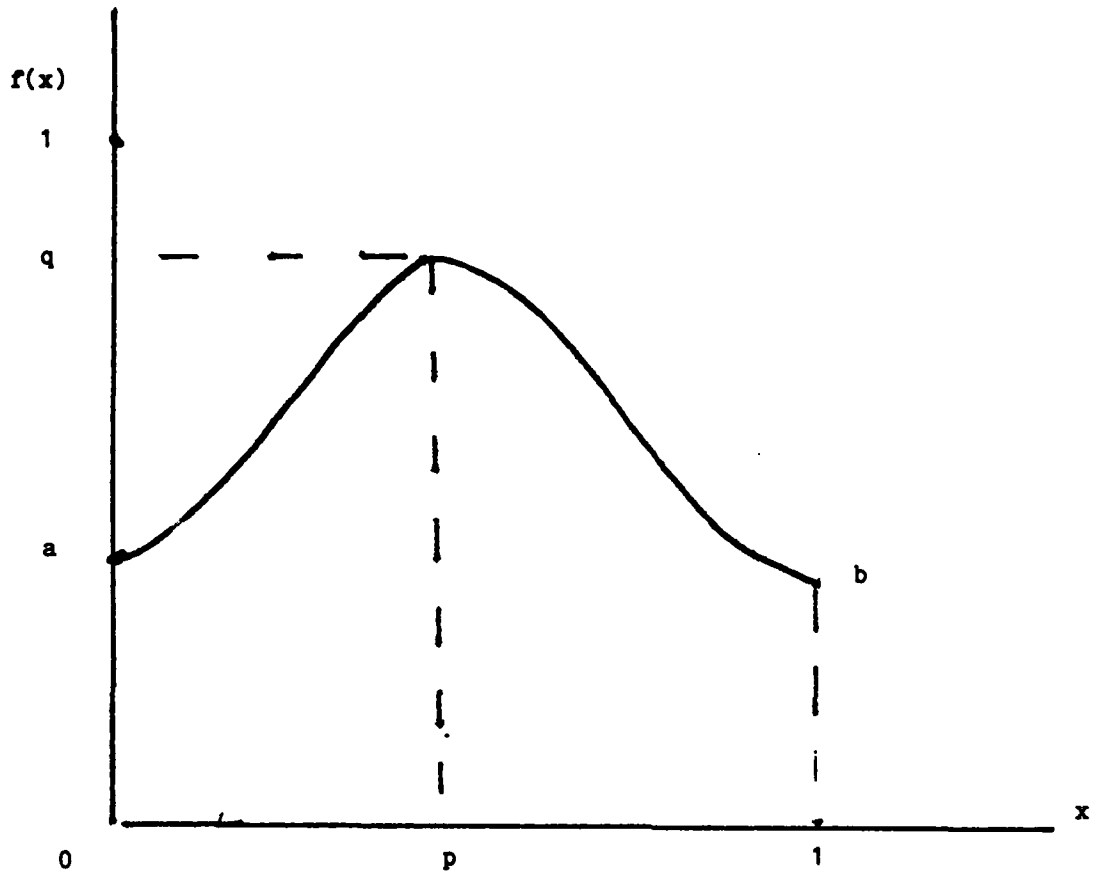


FIGURE 1

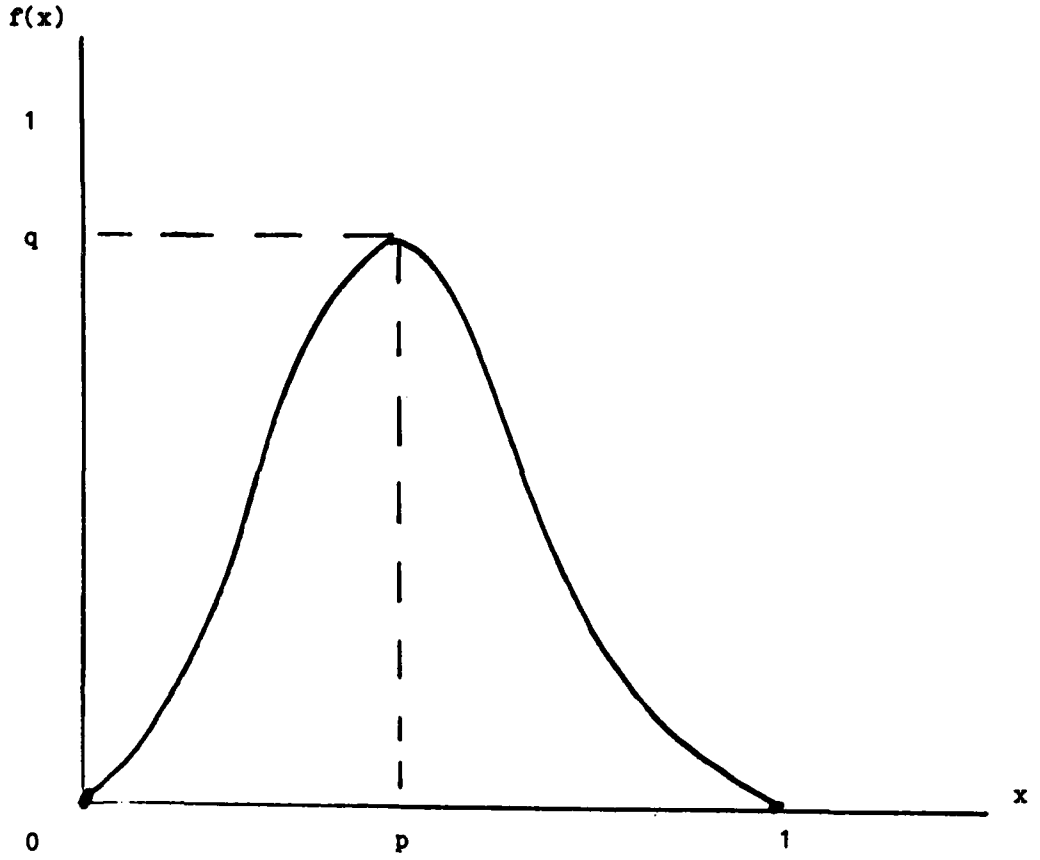


FIGURE 2

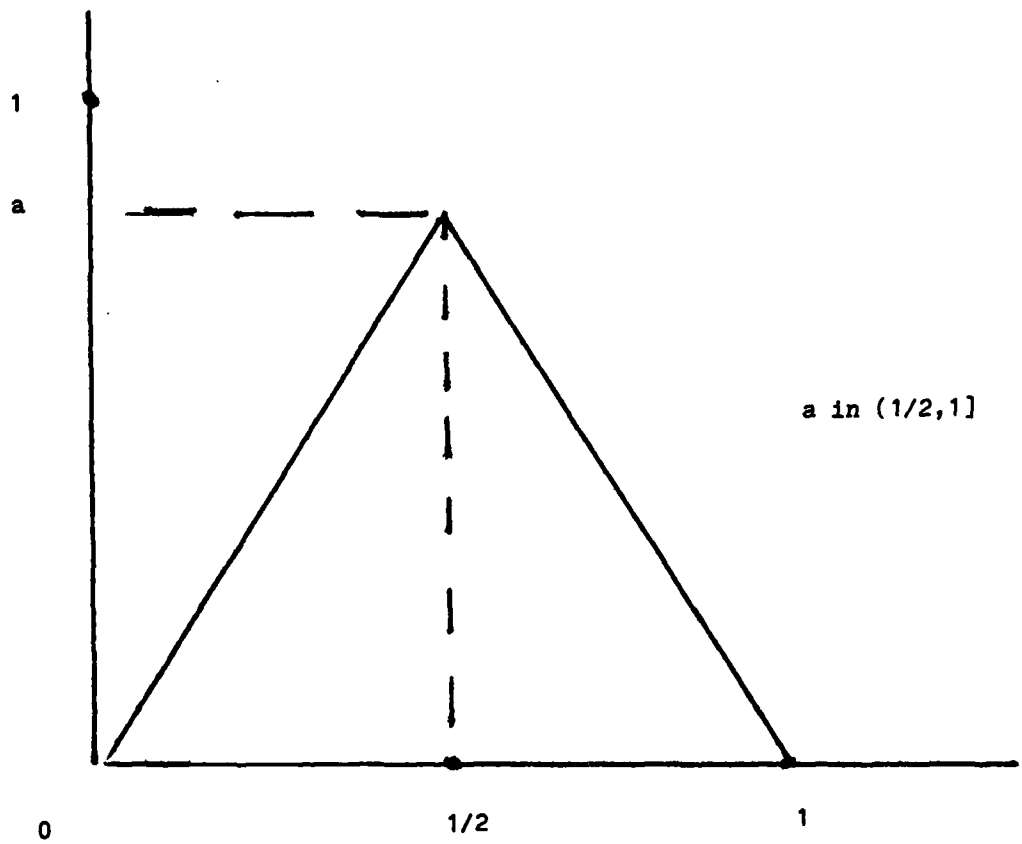


FIGURE 3

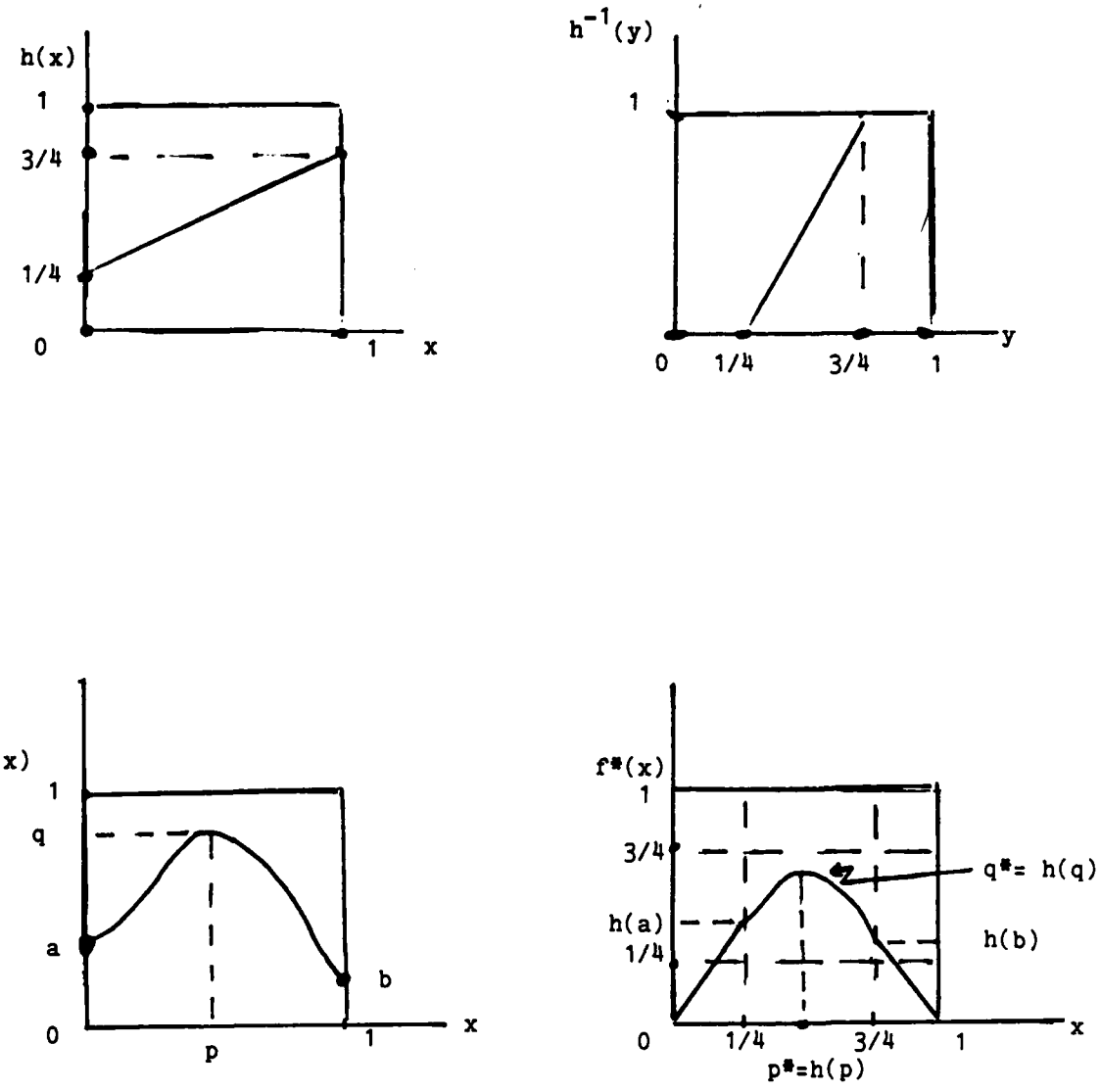

h(x)
1
3/4
1/4
0
1
x
$h^{-1}(y)$
1
0
1/4
3/4
1
y
f(x)
1
q
a
0
p
1
b
x
f*(x)
1
3/4
h(a)
1/4
0
1/4
3/4
1
x
q*= h(q)
h(b)
p*=h(p)

FIGURE 4

END

12-86

DTIC